UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Analyse statique de transformations pour l'élimination de motifs

THÈSE

présentée et soutenue publiquement le 08/09/2022

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Pierre LERMUSIAUX

Composition du jury

Présidente :	Maribel FERNANDEZ	
Rapporteurs :	Thomas GENET,	Professeur, Université de Rennes/IRISA
	Olga KOUCHNARENKO,	Professeure, Université de Franche-Comté/FEMTO
Examinateurs :	Maribel FERNANDEZ,	Professeure, King's College
	Marc PANTEL,	Maitre de Conférence, ENSEEIHT/IRIT
Encadrants :	Pierre-Étienne MOREAU,	Professeur, Université de Lorraine/LORIA
	Horatiu CIRSTEA,	Professeur, Université de Lorraine/LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Cette thèse est l'aboutissement d'un long travail qui n'aurait pas pu être possible sans le support, moral et/ou scientifique, de nombreuses personnes. Je tiens donc à remercier toutes les personnes qui m'ont aidé et m'ont permis d'arriver au bout de cette aventure, en commençant par mes encadrants Pierre-Étienne Moreau et Horatiu Cirstea. Pierre-Étienne m'a ouvert les portes de la recherche en me guidant dans mon cursus et en m'offrant cette opportunité unique de venir faire une thèse avec lui. Horatiu a été un support indéfectible tout au long de cette thèse, tant bien en physique qu'en distanciel, dans les moments difficiles comme dans les bons.

Un grand merci également à mes rapporteurs, Olga Kouchnarenko et Thomas Genet, pour le temps et l'intérêt qu'ils ont dédiés à mes travaux, ainsi que pour leurs retours et les échanges qui m'ont permis d'enrichir mon approche et mes raisonnements, et de peaufiner ce manuscrit. Merci à Maribel Fernandez qui, en tant que présidente du jury, a proposé un questionnement très enrichissant des perspectives possibles de mes travaux, et à Marc Pantel qui, aussi bien au sein de mon jury que dans le projet FORMEDICIS, a également contribué, par ces retours, de façon non-négligeable à mes travaux.

L'ensemble de l'équipe Véridis/Mosel m'a accueilli et soutenu tout au long de ma thèse. Merci Marie, Stephan, Dominique, Étienne, Pascal, Sophie, Engel... Je remercie en particulier mes co-bureaux ; Margaux et Daniel pour m'avoir montré le chemin, Hans pour la diversité incroyable de sujets de conversation qu'il a pu apporter au bureau (et Athénaïs pour m'avoir soutenu aussi bien dans mes travaux de thèse qu'une raquette de badminton à la main), et ceux qui me suivront (ou prendront leur propre chemin) : Rosalie, Dylan et Thomas. Un grand merci aux membres du projet ANR FORMEDICIS, et en particulier à David Chemouil pour avoir proposer l'inspiration initiale de mes travaux de thèse. Un dernier merci à Christophe Ringeissen qui, en tant que référent scientifique, a montré un intérêt manifeste dans mes travaux et m'a apporté de nombreux conseils au fur et à mesure de ma thèse.

Je tiens également à remercier toute ma famille et mes amis, qui m'ont apporté soutien et réconfort pendant cette thèse. Merci en particulier à mes parents (et beaux-parents) et à mon frère. J'ai également une petite pensée pour mes grands parents, dont seule ma Mamie a pu assister à ma soutenance, mais qui m'ont chacun inculqué des valeurs qui ont été essentielles durant ce doctorat. Merci à Jean-Charles et Charles pour m'avoir donné à la fois le pire et le meilleur conseil qu'ils pouvaient me donner (ils sauront de quoi je parle). Merci à Stéphane, à Luc et Camille, à Nicolas et Mélanie, à Claire et Alexandre, à Quentin et Malorie, à Julien, à Laurent et à Mikaël.

Table des matières

Table des figures

Extended abstract

Program transformations are a common practice in computer science, which dates back to the conception of the very first programming languages. Indeed, each language presents a syntax that is used to describe sequences of instructions thanks to a semantics. This conjunction of syntax and semantics is also supposed to make the language more intelligible than the machine code that allows its execution. In order to be able to run the actual program, its source code must then be compiled, *i.e.* interpreted and transformed into an executable equivalent.

With the growing popularity of information technologies, practices based on program transformations have become more and more prevalent. The integration of these technologies to critical systems, in particular in the aerospace and transport industries, has notably prompted the conception of techniques allowing to ensure the reliability and safety of the programs considered. Beyond making a program executable, program transformation techniques have evolved in order to provide tools that can guarantee, with a varying degree of rigorousness, that the behaviour of a program satisfies some predefined specifications.

Formally verifying that a program will run without any error is not only a very complicated problem, but the complexity of this problem generally grows exponentially with the complexity of the program itself. Therefore, techniques such as tests and simulations, that can provide some more limited guarantees of the soundness of a program may often be preferred. In the context of this kind of empirical approaches, program transformation approaches have been very useful to design automatic test generation tools or analysis tools that can identify *code-smells* or *anti-patterns*. But while these techniques have proved to be particularly useful to spot many errors and bugs, they can't formally guarantee that the program will run without error.

Therefore, outside of this kind of empirical approach, a lot of efforts have been made to design tools in order to provide some formal guarantee on the behaviour of a program. Such tools include proof assistants like Coq or Isabelle, but also many static analysis techniques allowing the formal verification of programs. In this context, program transformation techniques have also been indispensable, in particular in order to study program semantics, but also to design dedicated languages, such as specification languages.

From compilation to test generation, and including many analysis approaches, as well as formal verification techniques, program transformations are processes both ubiquitous and crucial to the proper functioning of programs and information systems. It is, therefore, essential to also be able to express and verify guarantees for these transformations.

We propose, in this thesis, to define a formalism allowing both the specification and verification of program transformations. In particular, we will focus on syntactical properties of the result of such transformation. In order to do so, we propose to use an algebraic approach based on the notion of rewriting.

Rewriting is a very common formalism, notably used in many approaches of formal verification, designed to express program transformations and describe their process. In this formalism, a transformation is described through a set of rewriting rules that provide a system of fundamental

algebraic specifications. Thanks to this algebraic approach, rewriting methods are particularly well suited to describe and propose an implementation of these transformations, while its formal aspects allow the study of many properties of a transformation, such as its termination and the shape ultimately obtained as the result of the transformation.

Many languages such as Tom [BBK+07], Maude [CELM96] or Stratego [Vis01], have, thus, used this rewriting formalism to present tools to design and analyse transformations. Its numerous qualities, and its popularity, have also made it a staple formal verification technique. For example, in [MB04], the rewriting calculi is used to describe the semantics of a programming language. This approach is not only interesting thanks to its ability to provide an executable definition of the language, but also thanks to the possibility to use this definition to propose some formal analysis of the programs. Other techniques, such as automata completion [Gen14], use rewriting in order to provide a description of the possible results obtained through a given transformation. These kinds of descriptions are particularly useful to prove many correctness properties of programs [FGT04, BGJR07].

The approach proposed in this thesis is more particularly inspired by works in the field of compilation. The conventional approach used by most of the current compilers strives to organize the transformation from the source code to the executable program as a sequence of transformation steps, called compilation passes. Each pass performs a transformation between intermediate representations of the compiled program. Originally, compilers used to restrain as much as possible the number of passes to optimize the compiler itself. Nowadays, the evolution of hardware provides compilers with much more memory and processing power, which allows them to increase the number of passes in order to produce a more optimized executable.

This multiplication in the number of compilation passes was naturally accompanied with a simplification of the behaviour of each pass. This allows to more precisely deal with the complexity of modern compilers, by breaking down complex operations into as small and as many passes as possible. Such passes usually perform a transformation that only impacts a small number of constructions of the intermediate languages considered. For example, de-sugaring passes will simply eliminate the constructions associated with the respective syntactic sugar, while optimization passes will focus on improving the behaviour of particular instruction, or sequence of instructions.

Each pass can thus be provided with a description of the intermediate input and output languages. Such a description not only provides a specification of the behaviour of each pass, but can also be verified to guarantee some correctness properties, with regards to these specifications, of the transformation performed by the compilation pass. As modern languages can easily contain dozens or even hundreds of different constructions, implementing the mechanisms of each single pass, as well as providing the descriptions of each intermediate language can be quite tedious and, generally, very redundant. Therefore, designing such a compiler can be both expensive and time-consuming.

In order to reduce a lot of boilerplate associated to the implementation of such passes, the Nanopass [KD13] library proposes to use automatic code generation tools to produce most of the redundant code, while relying on the type checking system of the Scheme language to provide syntactic guarantees regarding the specifications of input and output languages of each pass. As a compiler writing tool, Nanopass as seen applications both in an educational [SWD04] and in commercial [KD13] context as a compiler for Scheme, as well as an implementation in OCaml [Mil17].

However, in terms of static analysis, the Nanopass approach is a bit shallow, as it can only verify or contradict the existence of specific language symbols in the results returned by the transformation. In a more general context, we would like to be able to provide a more precise

description of these results, by considering patterns of these symbols.

This thesis presents a formalism and an analysis method, based on the rewriting formalism, to express and guarantee syntactic properties on the behaviour and the result of a transformation. To be more precise, the aim is to provide some correctness properties for program transformations by verifying that the result of such transformation cannot contain a given pattern. The approach relies on a notion of type annotations [FP91, RKJ08], in order to express the desired properties, as well as the necessary invariants. Moreover, by working with the rewriting formalism to describe the transformation in a term algebra, the approach considered is entirely algebraic, which also allows the use of pattern semantics [CM19] to represent the set of potential normal forms of the rewriting relation in a finite structure.

Pattern-free properties and semantics

The proposed formalism describes the desired guarantees using the pattern-free property, which states that the term considered must not contain a sub-term matched by a given pattern. This notion is used to define a system of annotations on the transformation function symbols : a function symbol $\varphi : s_1 * \cdots * s_n \mapsto s$ can be annotated by one or more profiles of the form $p_1 * \cdots * p_n \mapsto p$ where $p_1, \ldots, p_n, p$ are (disjunctions of) patterns, to specify that all reducts of a term headed by φ, with sub-terms respectively $p_1, \ldots, p_n$-free, must be p-free. The left-hand side of such a profile can, thus, be seen as a pre-condition, that needs to be verified to guarantee the post-condition expressed by the right-hand side.

The formal Definition 3.3 of the pattern-free property for general terms of the algebra considers a potential infinite number of values that can be potential normal forms of instances of the studied terms. In order, to reason over these infinite numbers of values, we rely on a generalization of the ground semantics introduced in [CM19] that provides finite structures representing the set of values matched by a given pattern. In our formalism, we propose to use the chosen annotations of the transformation function symbols to define a similar notion of ground semantics (Definition 3.4), that represents an over-approximation of the expected normal forms of a given term. As for the original notion of ground semantics, the one proposed here allow us to easily express and compute combinators (equivalent to set disjunction, conjunction or difference) over these semantics.

Furthermore, by considering the sub-term closure of this new notion of semantics, called deep semantics (Definition 3.12), we can give a necessary and sufficient condition of pattern-free properties (Proposition 3.9), stating that a term t is p-free if and only if the intersection of its deep semantics with the ground semantics of p is empty. Relying on the semantics combinators, this equivalence provides the basis of the systematic method introduced later to check pattern-free properties.

However, as both notions of pattern-freeness and generalized semantics rely on the symbol annotations, the formalism described assumes thus a specific shape for the potential normal forms of reducible terms. This assumption should be checked by verifying that that the chosen annotations are consistent with the rewriting system describing the behaviour of the considered transformation : any reduction, through the rewriting relation defined by the rewriting system, that leads to a term contradicting the expected pattern-free properties specified by the annotations in the original term would refute the chosen annotations of the transformation function symbols.

In other words, the profiles chosen to annotate the transformation symbols describe a specification, that we propose to check by verifying that the rewriting relation induced by a given rewriting system preserves the semantics of the transformed terms, thus guaranteeing that no

contradicting reduction exists. In order to prove the semantics preservation of the rewriting relation, we will show that we only need to check that each rule of the rewriting system satisfies each profile annotating the head symbol of its left-hand side : *i.e.* that for all substitutions such that the instances of the sub-terms in the left-hand side of the rule verify the pre-condition of the profile, the corresponding instance of the right-hand side must verify the post-condition (Proposition 3.18). Once again, relying on semantics combinators, we can thus introduce a systematic method of static analysis to verify these properties.

Static analysis of pattern-free properties

Therefore, the proposed static analysis tries to check that each rule of the considered rewriting system satisfies each profile of the annotation of the head-symbol of the left-hand side of the rule. For each rule of the rewriting system, and each profile considered, the analysis method is composed of three major steps :

1. the first step relies on a pattern aliasing approach and the Proposition 4.15 to propose an inference on substitutions verifying the pre-condition of the considered profile ;
2. the second step uses the notion of semantics equivalent (Definition 4.7) to describe the shape of the corresponding instances of the right-hand side of the studied rule ;
3. the last step proves the desired pattern-free properties of the inferred right-hand side of the rule, by checking that the semantics intersection considered by Proposition 3.9 is indeed empty.

In order to implement these analysis steps, it was first necessary to introduce and formally prove several mechanisms.

Deep semantics decomposition

Being able to study deep semantics is the first fundamental mechanism necessary to the analysis method proposed, as it is required to verify pattern-free properties. To do so, we propose in Section 4.1 an approach, relying on a graph representation of the semantics of the sub-terms considered, that can be used to check that the deep semantics of an extended pattern is not empty, and, in this case, decompose it as a union of ground semantics.

Thanks to this representation, we introduce the `getReachable` algorithm (Figure 4.7), in order to compute this decomposition. In the context of the overall analysis method, this algorithm can then be used to simplify the verification of pattern-free properties, by expressing the semantics intersection considered by Proposition 3.9 as conjunctions of patterns.

Reduction of pattern conjunction

In order to check that the semantics of these conjunctions are indeed empty, we propose a generalization of the approach presented in [CM19] to compute complements of patterns. This generalization relies on the rewriting system $\mathfrak{R}$, presented in Figure 4.8, to reduce conjunction of patterns while preserving their semantics.

From the decomposition of the deep semantics previously obtained, we can thus use the system $\mathfrak{R}$ to prove that the semantics intersection considered by Proposition 3.9 is indeed empty. These two mechanisms combined therefore provide a systematic method to prove pattern-free properties.

Inference of instantiation constraints

The overall analysis method relies on the notion of profile satisfaction to prove the consistency

of the chosen annotations. This notion states that whenever the pre-condition of a profile, as described by its left-hand side, is verified by an instance of the left-hand side of the rule, the post-condition, described by the right-hand side of the profile, should be verified by the corresponding instance of the right-hand side of the rule. In order, to prove such implication, we propose a mechanism to study and compute the instantiation constraints imposed by the pre-condition of a given profile.

The proposed method relies on an aliasing approach, and the system $\mathfrak{R}$ previously mentioned, to express the instantiation constraints associated to each variable of the left-hand side of the considered rewriting rule as extended patterns. The corresponding instances of the right-hand side of the rule can then be expressed thanks to a similar aliasing of the variables in this right-hand side, as proven in Proposition 4.15.

Semantics equivalent
However, the terms obtained by this inference mechanism can contain defined symbols, that make computing the semantics (deep and ground) of these terms particularly complicated. According to the definition of our generalized semantics, for each defined symbol, the resulting semantics of corresponding sub-terms is over-approximated based on the annotations of the symbol and the pattern-free properties of its own sub-terms. We thus propose to use the notion of semantics equivalent (Definitions 3.13 and 4.7) to build an equivalent pattern without any such symbol (Proposition 4.16).

The previously mentioned mechanisms not only allow the computation of this semantics equivalent, but they can then be used to verify that the resulting semantics intersections are indeed empty. All these mechanisms combined can thus be used to prove the semantics preservation of a given rewriting system, following the three steps described before.

Most of these mechanisms rely on a linearity hypothesis of the right-hand sides of the rules of the rewriting system analysed (*i.e.* right-hand sides where all variables can only occur once). However, the semantics approach proposed can actually take into account correlation constraints of the multiple instances of a given variable. The aliasing mechanism proposed for the inference step is indeed equally suited to study these constraints : such a correlation constraint is simply an instantiation constraint requiring a unique instance for different occurrences of a given variable. We then propose to show how these mechanisms can be adapted to analyze non-linear rewriting systems.

Static analysis of non-linear rewriting systems

The analysis method presented aims at verifying that the specifications given as annotations of the function symbols are verified by the rewriting relation induced by a rewriting system describing the behaviour of the corresponding transformation functions. The proposed approach for this analysis uses the equivalence between the notions semantics preservation and profile satisfaction. As such, it is highly reliant on a fundamental property of our generalized semantics : the semantics preservation by substitution, *i.e.* the ground semantics of any instance of a term should be a subset of the semantics of that term. However, this property is not guaranteed for non-linear terms, and therefore the proposed static analysis can only be applied to non-linear rewriting systems with a few restrictions.

In the context of a non-linear rewriting system, the aliasing mechanism, that is used to infer instantiation constraints, is indeed hindered by the lose of the preservation by substitution of non-linear terms. In order to apply the previously presented analysis method nevertheless, we

propose two possible strategies :

- an analysis by linearization of the inferred right-hand side : as the generalized ground semantics introduced in our formalism describes an over-approximation of the potential normal forms of reducible terms, it is possible to ignore any correlation constraints between different occurrences of a given variable by linearizing the inferred right-hand sides. While the over-approximation will obviously be less precise, there are many cases where the correlation constraints are unnecessary to prove the desired specifications, and this approach can thus still be sufficient to verify them.
- an analysis of rewriting relations under a strict reduction strategy : in term of the rewriting relation, assuming a strict reduction strategy is equivalent to only considering substitutions that instantiate variables with values, and such substitutions will still preserve the semantics of non-linear terms. Therefore, if we consider a transformation that satisfies this assumption, it is possible to use the analysis method as previously presented to verify the specifications described by the chosen annotations, while still taking into account correlation constraints between different occurrences of a given variable.

These two approaches still have non-negligible limitations. In both cases, as the generalized semantics chosen in our formalism represent an over-approximation of the potential normal forms of reducible terms, it can lead to some false negative results of the analysis. This is particularly the case for the analysis by linearization, as linearizing the terms leads to a rougher over-approximation. On the other hand, the strict reduction strategy assumption would limit the application of the static analysis to programs written in languages with *Call by value* evaluation strategies. In order to circumvent these limitations, we study in Section 5.2 other forms of semantics that would allow, under a confluence assumption (that is generally less limiting, particularly for the analysis of functional programs), a more precise over-approximation of the potential normal forms, and a better evaluation of the correlation constraints of variables.

The notion of confluence guarantees that two identical terms will ultimately be reduced to the same normal-form if they have one, and is generally an acceptable assumption in the context of static analysis of functional language programs. Therefore, assuming the confluence of the rewriting relation allows us to establish correlation constraints between two identical subterms. Among the different notions of semantics studied, we chose to work with a semantics over-approximating potential normal forms of a term by evaluating the term using substitutions instantiating variables with values. Thanks to the confluence assumption this substitutive semantics can, indeed, correctly evaluate such correlation constraints between two identical subterms. Moreover, unlike the generalized ground semantics introduced previously, it is preserved by substitution even for non-linear terms, but still allows us to establish an equivalence between semantics preservation of the rewriting relation induced by a rewriting system and the profile satisfaction of the rule of this system (Proposition 5.19), even if it is non-linear.

Similarly to the linear case, we can then propose a static analysis method, that aims to verify that each rule of the considered rewriting system satisfies each profile of the annotation of the head symbol of its left-hand side. Among the mechanism introduced for the linear analysis, the aliasing approach for the inference of instantiation constraints, the deep semantics decomposition, and the pattern conjunction reduction can still be used in the non-linear case, while the construction semantics equivalent, used to simplify the semantics of the inferred right-hand side, breaks correlation constraints of identical sub-terms and, thus, cannot be used anymore. For each rule of the rewriting system, and each profile considered, the non-linear analysis method is, therefore, only composed of two major steps :

1. the first step relies on a pattern aliasing approach and the Proposition 5.21 to propose an

inference of substitutions verifying the pre-condition of the considered profile ;

2. the second and final step uses a decision algorithm presented in Figure 5.1, that tries to build an evaluation context in which the post-condition of the profile would not be verified. If no such context exists, the algorithm allows us to conclude that the inferred right-hand side verifies the post-condition, and therefore that the rule satisfies the profile.

Under the confluence assumption, we can therefore provide a static analysis method to verify that the specifications, given as annotations of the function symbols are verified, by the rewriting relation induced by a (non-linear) rewriting system, describing the behaviour of the corresponding transformation functions.

Implementation, results and benchmarks

The static analysis method has been implemented in Haskell, and the implementation is publicly available and usable through an online interface. A test suite of hand written examples, from scenarios presented in Appendix C, is provided with the implementation.

Overall, the implementation follows closely the different mechanisms presented previously. One noteworthy difference concerns the `getReachable` algorithm, who is implemented by the program presented in Figure 6.1. This implemented version considers the same graph representation of pattern-free properties, but instead of first building the graph, and then analyzing it, it tries to do both simultaneously. This version performs, on average, about twice as fast as the one presented previously in Figure 4.7, while also taking into account potentially infinite terms described by the considered graph, which can be particularly interesting for the analysis of programs with a *lazy* evaluation strategy. The implementation also uses a caching approach to limit redundant computing of similar pattern-free properties. On average, this reduces the execution time of the static analyser by a factor of 2.

The implementation leaves the choice of using a linear, strict or non-linear analysing method to the user. By default, it implements an analysis in order to decide if the use of the non-linear approach is necessary, and if not runs a strict analysis. We also discuss the complexity of the analysis methods : the complexity is linear with the number of rules of the considered rewriting system, polynomial with the number of symbols in the left-hand side of each rule, and linear (on average) with the number of symbols in the right-hand side. The patterns chosen in the profile annotations have a polynomial effect on the complexity with regards to their number of symbols, and an exponential one with regards to their number of disjunctions. The big difference, in term of complexity, between the linear approach and the non-linear one concerns the number of profiles in annotations, which only have a polynomial effect in the linear analysis but an exponential one in the non-linear analysis. Moreover, the different polynomial effects are, generally, less favorably weighted for the non-linear analysis.

Finally, the implementation provides quite encouraging results when compared to other approaches of the literature. In the context of the analysis of program transformations, our annotation formalism is non-intrusive and rather intuitive, thanks to the algebraic aspects the pattern-matching properties considered. In the context of the analysis of functional programs, some approaches rely on a representation of terms using tree automata, which is more expressive than our formalism, but also harder to grasp for the non-expert user. In more general contexts, alternative analysis approaches tend to propose formalism more suited to the verification of programs handling integers and container datatypes, for whom they provide a wider expressiveness.

Future work

Several perspectives are considered to improve both our formalism and the analysis method proposed.

Expressiveness
One key weakness of our statical analysis approach, relies in the expressiveness of our formalism. While it is well suited in the context program transformations, our formalism relies on the notion of pattern-free properties to specify algebraic specifications of the analysed programs, and, therefore, cannot express or verify properties that cannot be described by the latter (as, for example, arithmetical properties on integers, size constraints on lists, ...).

In practice, our pattern semantics and most mechanisms proposed to implement our analysis method could be adapted to handle more general forms of algebraic annotations. We thus hope to be able to extend our annotation formalism, by using profiles that could implement notions of sub-typing similar to those proposed by CDuce [FCB02]. Such semantics sub-typing would, indeed, remain compatible with our algebraic and semantics formalism, while allowing our annotations to express more general specifications.

Another improvement path would be to study how the proposed analysis could handle higher-order functions. In the formalism presented in this thesis, we chose to restrict ourself to first-order programs, but higher-order functions are a staple feature of functional programming languages. Moreover, defining profiles on higher-order functions would allow our approach to express new form of guarantees that can probably not be expressed in its current state.

Profile Inference
While our annotation approach is rather non-intrusive, and quite intuitive for users familiar with functional programming, it remains quite cumbersome when compared to other, more automated, approaches of static analysis. We hope to be able, in the future, to implement an inference mechanism in order to limit the annotation burden assigned to the user, by deriving some of the profiles from a minimal specification and the analysed rewriting system.

Finally, we would like to study how the presented method can be used to implement static analysis on transformations using rewriting with strategies. In particular, using the encoding proposed in [CLM15], rewriting strategies can be expressed through a classical rewriting system. An inference mechanism would then be able to derive the profiles annotating the function symbols generated by the encoding, in order to apply our static analysis method. Our hope is that this could lead to an integration of the work presented in this thesis in languages that implement such strategical rewriting, like Tom.

Outline of the thesis

The thesis comprises 6 chapters. The implementation of both the linear analysis and the non-linear analysis is deliberately presented in a separate chapter from the respective presentation of the methods, in order to make comparisons between the two, in particular in terms of complexity. To help along the reading, the main body of the thesis only contains proofs that are considered fundamental to its overall comprehension. Proof of intermediate Lemmas can be found in Appendix A.

- In Chapter 1, we present preliminary notions and notations necessary to read the rest of the thesis.

- In Chapter 2, we introduce the problem considered in this thesis, inspired from the compilation pass approach proposed by the Nanopass library. We make an overview of languages based on the rewriting formalism, with a particular attention on the notion of rewriting strategies, that does answer some aspects of the considered problem. We conclude this Chapter with a presentation of other static analysis techniques, from the literature, that may be able to establish guarantees similar to those required by our problem.
- In Chapter 3, we present our formalism by defining notions of pattern-free properties and pattern semantics, using annotations on function symbols. We study the formal properties of the introduced notions, and more specifically their behaviour with regards to rewriting.
- In Chapter 4, we present a statical analysis method to verify that a considered rewriting system respects the behaviour specified by the chosen annotations. To do this, we propose several tools used to study and compare semantics, and a mechanism used to infer instantiation constraints of patterns.
- In Chapter 5, we study the application of the analysis method, presented in the previous Chapter, to non-linear rewriting systems. Taking into account the limitations that the previous method thus displays, we propose a formalism more suited to study pattern-free properties for non-linear patterns. We also present some adjustments of our analysis method, relying on this revised formalism.
- In Chapter 6, we present the implementation of the static analysis method(s) presented, and discuss the respective complexities of the linear and non-linear approaches. We detail some implementation choices, and propose optimizations, while presenting the results of our implementation. Finally, we compare these results with the other techniques presented in Chapter 2.

Introduction

Cette thèse propose une étude formelle des procédures de transformation de programmes basée sur la notion de réécriture. Plus précisément, elle présente un formalisme et une méthode d'analyse permettant d'exprimer et de garantir des propriétés syntaxiques sur le comportement et les résultats d'une telle transformation.

La transformation de programmes est en effet une pratique très courante dans le domaine des sciences informatiques, dont l'apparition renvoie directement à la conception des premiers langages de programmation. Chaque langage propose une syntaxe permettant d'écrire des séries d'opérations à l'aide d'une sémantique plus compréhensive que le langage machine dans lequel il est exécuté. Le code source d'un programme doit, pour cela, être compilé, i.e. interprété et transformé afin d'être rendu exécutable.

Par la suite, avec la démocratisation des technologies informatiques, les pratiques basées sur la transformation de programmes prennent une place de plus en plus importante. L'intégration de ces technologies à des systèmes critiques encourage notamment le développement de techniques et de méthodes permettant de garantir la fiabilité et la sureté des programmes utilisés. Au-delà du simple besoin d'exécuter un programme, les méthodes de transformation de langage évoluent ainsi pour fournir des outils permettant de garantir, plus ou moins formellement, le comportement d'un programme par rapport à un ensemble de spécifications données.

Ces outils de validation proposent généralement deux types d'approches distinctes : une approche empirique, basée sur les tests, la simulation ou encore des critères pratiques d'écriture de programmes, ou une approche formelle, basée sur des assistants de preuve tels que Coq *ou* Isabelle *et des outils d'analyse statique permettant la vérification formelle des programmes. Dans le premier cas, les techniques de transformation de programmes ont permis le développement d'outils de génération de tests, ainsi que d'approches d'analyses permettant la détection de* code-smells *et d'*antipatterns*, qui référencent des mauvaises pratiques pouvant entrainer des erreurs. Tandis que dans le deuxième cas, elles sont essentielles à l'étude sémantique des langages de programmation, ainsi qu'à la conception de langages dédiés, tels que des langages de spécifications.*

De la compilation à la génération de tests en passant par de nombreuses approches d'analyse de code et de vérification formelle des programmes, la transformation de programmes est ainsi un procédé qui est à la fois omniprésent et crucial au bon fonctionnement des programmes et systèmes informatiques. Il est donc primordial de pouvoir exprimer et vérifier des garanties sur ces transformations.

La réécriture est un formalisme très souvent utilisé, notament dans le domaine de la vérification formelle, pour exprimer des transformations de programmes, ainsi que pour en étudier les mécanismes. Dans ce formalisme, une transformation se présente sous la forme d'un ensemble de règles de réécriture qui décomposent la transformation considérée sous la forme d'un système de spécifications algébriques élémentaires. Grâce à cette approche algébrique, le concept de réécriture est particulièrement adapté à la description et à l'implémentation de transformations, tandis que son aspect formel permet d'étudier de nombreuses propriétés des transformations considérées,

comme leur terminaison et la forme des résultats obtenus.

De nombreux langages et outils, tels que Tom [BBK+07], Maude [CELM96] ou Stratego [Vis01], se sont ainsi basés sur ce formalisme de réécriture pour proposer des méthodes d'analyse et/ou de transformation de programmes. C'est également un formalisme qui est très apprécié pour son exécutabilité, notamment grâce à des approches de réécriture stratégique, comme celles mises en place par les langages cités.

Ces nombreuses qualités et sa popularité en font ainsi un formalisme parfaitement adapté dans le domaine de la vérification formelle. Par exemple, dans [MB04], la logique de réécriture est utilisée pour décrire la sémantique d'un langage de programmation. Cette approche permet non seulement de donner une définition exécutable du langage étudié, mais également d'utiliser cette définition pour réaliser des analyses formelles de programmes. D'autres approches, comme la complétion d'automate [Gen14], se basent ainsi sur la réécriture pour essayer de donner une caractérisation des résultats de la transformation considérée. Obtenir une telle caractérisation permet en effet de vérifier de nombreuses propriétés de correction des programmes [FGT04, BGJR07].

De façon similaire, on propose donc de mettre au point une nouvelle méthode permettant de donner de telles garanties sur les résultats et le comportement d'une transformation considérée, en s'inspirant du modèle de passes de compilation et de la librairie Nanopass [KD13]. Dans le domaine de la compilation, l'approche conventionnelle de conception des compilateurs consiste à séquencer la transformation en étapes de transformation minimales, appelées passes. Pour faciliter le travail d'écriture de ces passes pouvant être nombreuses et, bien souvent, assez redondantes, la librairie Nanopass se repose sur des outils de génération automatique de code, tout en se servant du système de type du langage Scheme pour garantir syntaxiquement que la transformation spécifiée respecte les langages d'entrée et de sortie de chaque passe.

En pratique, l'approche de Nanopass n'apporte cependant que de faibles garanties, puisqu'elle ne permet que de spécifier et de vérifier l'absence de certains symboles du langage. La finalité de cette thèse est donc de proposer un formalisme permettant de spécifier des garanties syntaxiques similaires, précisant l'absence de constructions plus complexes, via un système d'annotation des symboles de fonction de transformation. De plus, on proposera une méthode d'analyse permettant de vérifier que le système de réécriture, exprimant le comportement de ces fonctions, satisfait en effet ces spécifications.

Contributions

Formalisme d'Exemption de Motif

Dans le domaine de la compilation, la transformation de programmes a pour but d'interpréter le code source fourni et de générer un fichier exécutable correspondant. Afin de maximiser la modularité des compilateurs, cette transformation est généralement décomposée en une série de passes. Chaque passe ayant une fonctionnalité précise et limitée, elle n'affecte en pratique qu'un petit nombre des constructions du langage considéré. Pour simplifier l'aspect chronophage et redondant de l'écriture de ce type de passes, la librairie de compilation Nanopass propose de simplifier l'écriture des passes, en générant automatiquement une partie du code de chaque passe, ainsi que leur langage d'entrée et de sortie. En particulier, elle permet de définir ces langages intermédiaires par différence, en identifiant les constructions éliminées du langage d'origine.

En s'inspirant de l'approche de Nanopass, on propose un formalisme d'annotation des fonctions de transformation afin de décrire un ensemble de spécifications algébriques du comportement de ces fonctions. L'approche considérée consiste à garantir que le résultat de cette transformation n'est composé d'aucun élément filtré par certains motifs, déterminés par le système d'annotations. Cette approche se repose sur la notion d'Exemption de Motif qui décrit dans une

algèbre de termes, l'absence de sous-termes filtrés par un motif. À partir de cette notion, on introduit un formalisme de sémantique permettant de sur-approximer les résultats potentiels de la transformation, comme spécifié par les annotations des symboles.

Pour vérifier que la transformation respecte ces spécifications algébriques, il est donc nécessaire de prouver que le résultat de la transformation est en accord avec cette sur-approximation. L'aspect algébrique des propriétés considérées fait de la réécriture le formalisme naturel à la vérification de ces propriétés. On étudie donc le comportement des notions d'Exemption de Motif et de sémantique par réduction suivant la relation induite par le système de réécriture encodant la transformation. En pratique, la propriété recherchée se traduit par la préservation de la sémantique par relation de réécriture, dont on proposera donc une méthode statique de vérification.

Méthode d'analyse statique

Avec l'omniprésence croissante des technologies informatiques dans notre société moderne, la question de confiance des programmes, et plus généralement des systèmes informatiques, est devenue une considération primordiale. Pour répondre à cette problématique, de nombreuses approches proposent des méthodes d'analyse permettant la certification des programmes. Outre leur niveau de formalisation, ces méthodes se classent généralement en deux catégories : les méthodes dynamiques, comme le test ou la simulation, qui requiert l'exécution du programme, et les méthodes statiques qui cherchent à décrire et vérifier le comportement du programme. Ces méthodes statiques peuvent être très variées, allant de la preuve de programme et du *model-checking* à l'utilisation d'outils spécifiques permettant la description et la vérification de spécifications dédiées à un domaine particulier.

Le formalisme d'Exemption de Motif proposant, par le biais d'un système d'annotations, une spécification du comportement des fonctions considérées, on présente donc une méthode d'analyse statique permettant de vérifier le respect de ces spécifications. Cette méthode se repose sur l'étude du système de réécriture encodant le comportement des fonctions, et plus particulièrement on vérifie que chaque règle du système préserve la sémantique. Pour chaque règle, la méthode d'analyse suit trois étapes :

1. une étape d'inférence qui, à partir du comportement spécifié par le système d'annotations, déduit les cas d'application de la règle étudiée ;
2. une étape de construction de sémantique qui, à partir des cas d'application obtenus et de l'annotation des symboles, reconstruit une sur-approximation des résultats obtenus par application de la règle ;
3. une étape de vérification, qui vérifie que cette sémantique exprimant la sur-approximation des résultats respecte les spécifications données par le système d'annotations.

Ces différentes étapes se reposent sur un certain nombre de mécanismes allant de la comparaison de sémantique de motifs, à l'étude intrinsèque des propriétés d'Exemption de Motif dans la signature de types considérée, en passant par l'inférence de contraintes d'application des règles de réécriture.

Dans certains cas, les systèmes de réécriture utilisent des règles dîtes non-linéaires, car elles mettent en jeu certaines contraintes de corrélation, et l'application de certains de ces mécanismes peut être limité par ces contraintes. On étudie donc plus spécifiquement l'application de la méthode d'analyse à ces systèmes non-linéaires, qui n'est possible qu'en sur-approximant le comportement de la relation réécriture par celle d'un système linéaire, ou en se restreignant à l'étude de programmes utilisant une stratégie d'évaluation en *appel par valeur*. Pour éviter ces limitations, on introduit un formalisme adapté permettant de prendre en compte de manière plus précise les contraintes de non-linéarité des systèmes. On propose ainsi une adaptation de la

méthode d'analyse statique utilisant ce formalisme pour l'étude des systèmes non-linéaires.

Implémentation
Une implémentation de la méthode d'analyse statique est publiquement disponible [CL20]. Cette implémentation est également testable directement en ligne, et une branche du dépôt est destinée aux tests de performance (principalement en termes de temps d'exécution). Un certain nombre de scénarios, présentés en Annexe, sont également fournis. On compare, en termes d'expressivité et de performance, les résultats obtenus par notre implémentation à d'autres approches de la littérature permettant de vérifier des garanties similaires.

Plan de la thèse

Ce manuscrit se décompose en 6 chapitres. Un choix a été fait de présenter l'implémentation et la complexité des méthodes d'analyse linéaire et non-linéaire dans un même chapitre afin de mieux les mettre en parallèle. Pour faciliter la lecture, on a également fait le choix de ne garder dans le corps du manuscrit que les preuves jugées essentielles à sa compréhension. Les preuves des Lemmes intermédiaires peuvent être retrouvées dans l'Annexe A.

- Dans le Chapitre 1, on présente les notions préliminaires et les notations essentielles à la lecture du reste du manuscrit.
- Dans le Chapitre 2, on introduit la problématique considérée dans cette thèse, à partir du modèle de passes de compilation proposé par la librairie Nanopass. On propose un tour d'horizon des langages à base de réécriture, avec une attention particulière donnée à la réécriture par stratégie qui répond à certains éléments de la problématique considérée par Nanopass. On conclue ce Chapitre par une présentation des approches d'analyse statique de littérature permettant d'établir des garanties similaires à celles considérées par notre problématique.
- Dans le Chapitre 3, on présente notre formalisme en introduisant une approche d'annotation des symboles et en définissant les notions d'Exemption de Motif et de sémantique de motifs. On étudie les propriétés théoriques de ces notions, et notamment leur comportement dans le contexte du formalisme de réécriture.
- Dans le Chapitre 4, on présente la méthode d'analyse statique permettant de vérifier que le système de réécriture considéré respecte le comportement spécifié par les annotations choisies. On propose pour cela des outils permettant l'étude approfondie des sémantiques, et leur comparaison, ainsi qu'un mécanisme d'inférence de contraintes d'instanciation d'un motif.
- Dans le Chapitre 5, on étudie l'application de la méthode d'analyse, proposée dans le Chapitre précédent, aux systèmes non-linéaires. Étant donné les limitations de cette méthode, on propose alors un formalisme adapté à l'étude des propriétés d'Exemption de Motif dans le cas non-linéaire. On présente également une adaptation de la méthode d'analyse se reposant sur ce formalisme.
- Dans le Chapitre 6, on présente l'implémentation et les complexités respectives des méthodes d'analyse linéaires et non-linéaires. On discute également certains choix d'implémentation et on propose certaines optimisations, en présentant les résultats obtenus. Enfin, on compare ces résultats à ceux obtenus par les approches présentées dans le Chapitre 2.

1

Notions Préliminaires

On présente dans ce Chapitre des notions de bases et les notations utilisées par la suite. Ces notions sont présentées de façon plus détaillée dans [BN98] et [Ter03].

1.1 Algèbres des termes

Nous définissons dans cette section les notions de base d'algèbres de termes du premier ordre.

Définition 1.1 (Signature)**.** *Une signature Σ consiste en un ensemble de symboles $\mathcal{F}$, tel que chaque symbole $f \in \mathcal{F}$ est associé à un entier naturel par la fonction d'arité, notée $arity : \mathcal{F} \mapsto \mathbb{N}$.*

On note $\mathcal{F}^n$ le sous-ensemble des symboles d'arité n. Les symboles de $\mathcal{F}^0$ sont appelés constantes.

On considèrera dans ce manuscrit des termes sortés, obtenus en donnant à chaque symbole une signature de sorte. On parle dans ce cas de signature multi-sortée.

Définition 1.2 (Signature multi-sortée)**.** *Une signature multi-sortée Σ consiste en un couple $(\mathcal{S}, \mathcal{F})$ où $\mathcal{S}$ est un ensemble de sortes et $\mathcal{F}$ est un ensemble de symboles, tel que chaque symbole $f \in \mathcal{F}$ dispose d'un* domaine $\mathcal{D}om(f) = s_1 * \cdots * s_n \in \mathcal{S}^*$ *et d'un* co-domaine $\mathcal{C}o\mathcal{D}om\,(f) = s \in \mathcal{S}$.

On note $\mathcal{F}_s$ le sous-ensemble des symboles de co-domaine s. Étant donné $f \in \mathcal{F}$, on note $f : s_1 * \cdots * s_n \mapsto s$ pour indiquer que $\mathcal{D}om(f) = s_1 * \cdots * s_n$ et $\mathcal{C}o\mathcal{D}om\,(f) = s$, et on pourra aussi noter f_s pour indiquer le co-domaine de f.

Dans le cas d'une signature multi-sortée, pour tout $f \in \mathcal{F}$ on a donc $arity\,(f) = |\mathcal{D}om(f)|$.

Dans le reste du manuscrit, on considérera des termes sortés, et on définit les notions suivantes uniquement dans le contexte d'une signature multi-sortée. Étant donnée une telle signature, on peut décrire les termes construits sur ses symboles et un ensemble dénombrable de variables sortées $\mathcal{X}$.

Définition 1.3 (Termes)**.** *Étant données une signature $\Sigma = (\mathcal{S}, \mathcal{F})$ et un ensemble dénombrable de variables sortées $\mathcal{X} = \bigcup_{s \in \mathcal{S}} \mathcal{X}_s$, où $\mathcal{X}_s$ désigne l'ensemble de variables de sorte s, on définit l'ensemble des termes de sorte $s \in \mathcal{S}$, noté $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$, comme le plus petit ensemble tel que :*

- *$\mathcal{X}_s \subseteq \mathcal{T}_s(\mathcal{F}, \mathcal{X})$, i.e. toute variable de $\mathcal{X}_s$ est un terme de $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$;*
- *pour tout symbole $f : s_1 * \cdots * s_n \mapsto s$ et pour tous $t_1, \ldots, t_n$ éléments respectifs de $\mathcal{T}_{s_1}(\mathcal{F}, \mathcal{X}), \ldots, \mathcal{T}_{s_n}(\mathcal{F}, \mathcal{X})$, le terme $f(t_1, \ldots, t_n)$ est un élément de $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$.*

L'ensemble $\mathcal{T}(\mathcal{F},\mathcal{X}) = \bigcup_{s\in\mathcal{S}} \mathcal{T}_s(\mathcal{F},\mathcal{X})$ *désigne donc l'ensemble des termes sortés.*

Pour $t \in \mathcal{T}_s(\mathcal{F},\mathcal{X})$*, on note* $t : s$ *pour indiquer que* t *est de sorte* s*. Étant donnée une variable* x *de sorte* s*, on pourra la noter* x_s *pour indiquer sa sorte.*

Étant donné un terme $t = f(t_1,\ldots,t_n)$, on dit que f est le symbole de tête de t. Pour les constantes, on confond le terme $a \in \mathcal{T}(\mathcal{F},\mathcal{X})$ et le symbole $a \in \mathcal{F}^0$.

Définition 1.4 (Variables)**.** *Soit* $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$*, on note* $\mathcal{V}ar(t)$ *l'ensemble des variables de* t*. Si* $\mathcal{V}ar(t)$ *est vide, on dit que* t *est un terme* clos*, et on note* $\mathcal{T}(\mathcal{F})$ *l'ensemble des termes clos.*

Dans le contexte des langages fonctionnels, on distingue les symboles de fonction, ou *symboles définis*, des *constructeurs*.

Définition 1.5 (Termes constructeurs)**.** *Étant données une signature* $\Sigma = (\mathcal{S},\mathcal{F})$*, on partitionne* $\mathcal{F}$ *en* $\mathcal{D}$*, l'ensemble des* symboles définis*, et* $\mathcal{C}$*, l'ensemble des* constructeurs*.*

On note donc $\mathcal{T}(\mathcal{C},\mathcal{X})$ *l'ensemble des termes constructeurs, et on appelle* valeur *tout terme clos constructeur, i.e. les termes de* $\mathcal{T}(\mathcal{C})$*.*

Pour des questions de lisibilité, on pourra par la suite noter les sortes d'une signature, et leurs constructeurs, sous la forme de types algébriques :

Exemple 1.1. *On définit la signature sortée* $\Sigma = (\mathcal{S},\mathcal{C} \uplus \mathcal{D})$ *décrit par les types algébriques :*

$$\begin{array}{rcl@{\qquad\qquad}rcl} S_1 & := & A & S_2 & := & c(S_1,S_2) \\ & | & B & & | & d(S_1,S_1) \end{array}$$

On a alors $\mathcal{S} = \{S_1,S_2\}$ *et* $\mathcal{C} = \{A : S_1, B : S_2, c : S_1 * S_2 \mapsto S_2, d : S_1 * S_1 \mapsto S_2\}$*.*

La notion de position permet d'identifier de façon unique les sous-termes.

Définition 1.6 (Position)**.** *Une position* ω *dans un terme* t *est une séquence d'entiers naturels décrivant le chemin de la racine de* t *à la racine du sous-terme à cette position, noté* $t_{|\omega}$*. La position vide, i.e. la position de la racine, est notée* ϵ*, et la concaténation de deux positions* ω_1 *et* ω_2 *est notée* $\omega_1.\omega_2$*.*

On note $\mathcal{P}os(t)$ *l'ensemble des positions de* t*.*

Étant donné deux termes t et t', et une position ω dans t, on note $t\,[t']_\omega$ le remplacement du sous-terme à la position ω par t' dans t. Pour des termes sortés, pour que $t\,[t']_\omega$ soit bien sorté, il faut donc que t' et $t_{|\omega}$ aient la même sorte.

Définition 1.7 (Préfixe)**.** *On note* $<$ *la relation naturelle de préfixe entre deux positions, i.e.* $\omega_2 < \omega_1$ *si et seulement si il existe* ω *telle que* $\omega_1 = \omega_2.\omega$*.*

Enfin, la propriété de linéarité qualifie les termes dont toutes les variables n'apparaissent qu'une fois.

Définition 1.8 (Linéarité)**.** *Un terme* t *est* linéaire *si et seulement si toute variable* $x \in \mathcal{V}ar(t)$ *n'apparaît qu'une fois dans* t*, i.e. il existe une unique position* $\omega \in \mathcal{P}os(t)$ *telle que* $t_{|\omega} = x$*.*

1.2 Filtrage et Sémantique de motif

Une substitution est une opération de remplacement définie par une fonction qui associe certaines variables à des termes de l'algèbre.

Définition 1.9 (Substitution)**.** *Une substitution σ est une fonction de $\mathcal{X}$ vers $\mathcal{T}(\mathcal{F},\mathcal{X})$, qui est l'identité sauf pour un sous-ensemble de $\mathcal{X}$ appelé* domaine *de σ et noté $\mathcal{D}om(\sigma)$. Quand $\mathcal{D}om(\sigma)$ est fini, on pourra l'expliciter sous la forme $\sigma = \{x_1 \mapsto t_1, \ldots, x_m, \mapsto t_m\}$.*
On étend en général la substitution σ à son endomorphisme sur $\mathcal{T}(\mathcal{F},\mathcal{X})$.

Étant donnée une substitution σ, on a donc :

- $\sigma(x) = \begin{cases} \sigma(x) & \text{si } x \in \mathcal{D}om(\sigma) \\ x & \text{sinon} \end{cases}$
- $\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$

Dans le cadre d'algèbres multi-sortées, on ne considère que des substitutions qui préservent la sorte d'un terme.

Définition 1.10 (Substitution sortée)**.** *On dit qu'une substitution σ est* sortée *si et seulement si elle n'associe des variables qu'avec des termes de même sorte, i.e. pour toute variable $x \in \mathcal{X}$, si $x : s$, alors $\sigma(x) : s$.*

Dans le contexte des langages fonctionnels, on pourra parfois expliciter le comportement d'une stratégie en *Appel par valeur* en utilisant des substitutions qui n'associent des variables qu'avec des valeurs.

Définition 1.11 (Substitution valeur)**.** *Une substitution ς qui n'associe des variables qu'avec des valeurs, i.e. telle que $\varsigma(x) \in \mathcal{T}(\mathcal{C})$ pour toute variable $x \in \mathcal{D}om(\varsigma)$, est appelée* substitution valeur.

La notion de substitution permet de définir le filtrage par motif.

Définition 1.12 (Filtrage)**.** *Soient un terme $p \in \mathcal{T}(\mathcal{F},\mathcal{X})$, appelé motif, et un terme clos $t \in \mathcal{T}(\mathcal{F})$, on dit que p filtre t, noté $p \lessdot t$, si et seulement s'il existe une substitution σ telle que $\sigma(p) = t$.*

$$p \lessdot t \iff \exists\, \sigma.\sigma(p) = t$$

Dans le cas de motifs linéaires, on peut également donner une définition opérationnelle du filtrage par motif :

$$\begin{array}{rcll} x_s & \lessdot & t & \iff t : s \quad \text{pour } x_s \in \mathcal{X} \\ f(p_1,\ldots,p_n) & \lessdot & f(t_1,\ldots,t_n) & \iff \wedge_{i=1}^{n} p_i \lessdot t_i \quad \text{pour } f \in \mathcal{F} \end{array}$$

Comme introduit dans [CM19], par la suite, on considèrera un filtrage par motif utilisant des motifs étendus incluant les opérateurs de disjonction + et de complément \, ainsi que le motif nul $\perp$ [1] :

1. On suppose, bien entendu, que $+, \setminus, \perp \notin \mathcal{F}$

Définition 1.13 (Motif étendu)**.** *Étant donné un ensemble de variables sortées $\mathcal{X}$ et une signature $\Sigma = (\mathcal{S}, \mathcal{D} \uplus \mathcal{C})$, l'ensemble $\mathcal{P}(\mathcal{C}, \mathcal{X})$ des* motifs étendus *est défini comme l'ensemble des termes de la forme :*

$$p, q \quad := \quad x \mid c(q_1, \ldots, q_n) \mid p_1 + p_2 \mid p_1 \setminus p_2 \mid \bot$$

avec $x \in \mathcal{X}, p, p_1, p_2 : s \in \mathcal{S}, c : s_1 * \cdots * s_n \mapsto s \in \mathcal{C} \cup \mathcal{N}$ et $\forall i \in [1, n], q_i : s_i$, où $\mathcal{N}$ représente l'ensemble des constructeurs de tuples notés $(\!|q_1, \ldots, q_n|\!)$.

Par extension, étant donné $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$ on note $\mathcal{V}ar(p)$ l'ensemble des variables de p.

Étant donné un n-uplet q, on notera généralement $q = (\!|q_1, \ldots, q_n|\!)$.

La notion de filtrage se généralise naturellement pour les motifs étendus, dans le cas de motifs linéaires. On définit donc la notion de linéarité pour les motifs étendus via une notion de méta-variable, représentant les variables restreintes par le filtrage :

Définition 1.14 (Linéarité)**.** *Étant donné un motif étendu $p \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, l'ensemble $\mathcal{MV}(p)$ des* méta-variables *du motif p est défini par :*

$$\begin{aligned} \mathcal{MV}(x) &= \{x\}, \textit{pour tout } x \in \mathcal{X} \\ \mathcal{MV}(c(p_1, \ldots, p_n)) &= \mathcal{MV}(p_1) \cup \ldots \cup \mathcal{MV}(p_n), \textit{pour tout } c \in \mathcal{C} \\ \mathcal{MV}(p_1 + p_2) &= \mathcal{MV}(p_1) \cup \mathcal{MV}(p_2) \\ \mathcal{MV}(p_1 \setminus p_2) &= \mathcal{MV}(p_1) \\ \mathcal{MV}(\bot) &= \emptyset \end{aligned}$$

Un motif étendu de la forme $c(p_1, \ldots, p_n)$ est linéaire si et seulement si chaque p_i, $i \in [1, n]$, est linéaire, et $\forall 1 \leq i < j \leq n$, $\mathcal{MV}(p_i) \cap \mathcal{MV}(p_j) = \emptyset$. Un motif étendu de la forme $p_1 + p_2$, resp. $p_1 \setminus p_2$, est linéaire si et seulement si p_1 et p_2 sont linéaires.

Intuitivement, pour que $c(p_1, \ldots, p_n)$ soit linéaire, les sous motifs $p_i, i \in [1, n]$ doivent être linéaires et deux à deux indépendants, tandis que pour $p_1 + p_2$, resp. $p_1 \setminus p_2$, p_1 et p_2 représente des alternatives indépendantes, donc leurs variables sont également indépendantes.

Exemple 1.2. *On considère des motifs de l'algèbre définie par la signature σ introduite dans l'Exemple 1.1 :*

- *Le motif $d(A, x) + d(x, B)$ est linéaire, puisque l'opérateur de disjonction $+$ rend les deux instances de la variable x indépendantes ;*
- *Le motif $d(x, B + x)$ est non-linéaire, puisque les deux instances de la variable x se trouvent sous le constructeur d et ne sont donc pas indépendantes ;*
- *Le motif $d(x, y \setminus x)$ est linéaire, puisqu'une des instances de la variable x se trouve à droite d'un opérateur complément $\setminus$, ce qui la rend indépendante.*

On pourra noter que ce dernier motif est donc équivalent à $d(x, y \setminus z)$ [1]*.*

On peut ainsi donner une définition opérationnelle du filtrage par motif étendu :

Définition 1.15 (Filtrage étendu)**.** *Soient un motif étendu linéaire $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$ et une valeur $v \in \mathcal{T}(\mathcal{C})$, on note $\prec\!\!\!\prec$ la relation de filtrage par motif étendu, définie telle que :*

$$\begin{aligned} x_s &\prec\!\!\!\prec v && \iff && v : s && \textit{pour } x_s \in \mathcal{X} \\ c(p_1, \ldots, p_n) &\prec\!\!\!\prec c(v_1, \ldots, v_n) && \iff && \wedge_{i=1}^{n} p_i \prec\!\!\!\prec v_i && \textit{pour } c \in \mathcal{C} \\ p_1 + p_2 &\prec\!\!\!\prec v && \iff && p_1 \prec\!\!\!\prec v \ \vee \ p_2 \prec\!\!\!\prec v \\ p_1 \setminus p_2 &\prec\!\!\!\prec v && \iff && p_1 \prec\!\!\!\prec v \ \wedge \ p_2 \not\prec\!\!\!\prec v \\ \bot &\not\prec\!\!\!\prec v && \iff && \end{aligned}$$

1. Et on peut obsever avec la définition du filtrage étendue que ce motif ne filtre aucune valeur, il est donc également équivalent à $\bot$.

On peut également intégrer la notion d'anti-motif [KKM07] au mécanisme de filtrage présenté :

Définition 1.16 (Anti-motif)**.** *Soit un motif étendu linéaire $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$ avec $p : s$, on appelle* anti-motif *de p, noté $!p$, le motif tel que $!p \nprec v$, pour tout $v \in \mathcal{T}_s(\mathcal{C}, \mathcal{X})$ avec $p \not\prec v$.*

Dans le formalisme de motif étendu présenté, l'anti-motif $!p$ peut donc être considéré comme un sucre syntaxique du complément $x_s \setminus p$.

1.3 Réécriture

Un système de réécriture est composé d'un ensemble de règles.

Définition 1.17 (Règle)**.** *Une règle de réécriture est un couple (l, r) de termes de $\mathcal{T}(\mathcal{F}, \mathcal{X})$, généralement notée $l \twoheadrightarrow r$, tels que que $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. On dit que l est le membre gauche de la règle et r le membre droit.*

Dans le contexte des langages fonctionnels, on peut utiliser un système de réécriture avec des règles de réécriture à constructeurs pour représenter le comportement d'une fonction.

Définition 1.18 (Règle de réécriture à constructeurs)**.** *Une règle de réécriture à constructeurs est une règle de réécriture $l \twoheadrightarrow r$, telle que l est de la forme $\varphi(l_1, \ldots, l_n)$ avec $\varphi \in \mathcal{D}^n$ et $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, pour tout $i \in [1, n]$.*

Définition 1.19 (Système de réécriture)**.** *Un système de réécriture (*TRS : Term Rewriting System*) $\mathcal{R}$ est un ensemble de règles de réécriture. Si toutes les règles de $\mathcal{R}$ sont des règles de réécriture à constructeurs, on dit que $\mathcal{R}$ est un système de réécriture à constructeurs (*CBTRS : Constructor based TRS*).*

Un système de réécriture induit une relation binaire de réécriture.

Définition 1.20 (Réécriture)**.** *Étant donné un système de réécriture $\mathcal{R}$, on définit la relation de réécriture sur $\mathcal{T}(\mathcal{F})$ telle qu'un terme $u \in \mathcal{T}(\mathcal{F})$ se réécrit en $v \in \mathcal{T}(\mathcal{F})$ dans le système $\mathcal{R}$, notée $u \Longrightarrow_{\mathcal{R}} v$, si et seulement s'il existe une règle $l \twoheadrightarrow r \in \mathcal{R}$, une position $\omega \in \mathcal{P}os(u)$ et une substitution σ telles que :*

- $u_{|\omega} = \sigma(l)$
- $v = u\,[\sigma(r)]_\omega$

Définition 1.21 (Fermeture)**.** *Étant donné un système de réécriture $\mathcal{R}$, on note $\Longrightarrow^*_{\mathcal{R}}$ la fermeture réflexive et transitive de $\Longrightarrow_{\mathcal{R}}$.*

Définition 1.22 (Formes normales)**.** *Étant donné un système de réécriture $\mathcal{R}$, un terme $t \in \mathcal{T}(\mathcal{F})$ est* réductible *dans le système $\mathcal{R}$ si et seulement il existe t' tel que $t \Longrightarrow_{\mathcal{R}} t'$.*

Un terme $v \in \mathcal{T}(\mathcal{F})$ non réductible est appelé forme normale*, et, étant donné un terme $t \in \mathcal{T}(\mathcal{F})$, on dit d'un terme $v \in \mathcal{T}(\mathcal{F})$ non réductible tel que $t \Longrightarrow^*_{\mathcal{R}} v$ est une forme normale de t.*

Les propriétés de terminaison et de confluence garantissent respectivement l'existence et l'unicité des formes normales dans le système de réécriture considéré.

Définition 1.23 (Terminaison)**.** *Étant donné un système de réécriture $\mathcal{R}$:*

- *la relation de réécriture est fortement terminante s'il n'existe pas de suite infinie* $(t_i)_{i\geq 1}$ *telle que* $t_1 \Longrightarrow_{\mathcal{R}} t_2 \Longrightarrow_{\mathcal{R}} \cdots$;
- *la relationde récriture est faiblement terminante si pour* $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$*, il existe* t' *non réductible tel que* $t \Longrightarrow^*_{\mathcal{R}} t'$.

Définition 1.24 (Confluence)**.** *Étant donné un système de réécriture* $\mathcal{R}$*, la relation de réécriture est confluente si et seulement si, pour tout termes* t, u, v *de* $\mathcal{T}(\mathcal{F})$*, on a :*

$$t \Longrightarrow^*_{\mathcal{R}} u \wedge t \Longrightarrow^*_{\mathcal{R}} v \implies \exists\, w \in \mathcal{T}(\mathcal{F}).u \Longrightarrow^*_{\mathcal{R}} w \wedge v \Longrightarrow^*_{\mathcal{R}} w$$

Pour un système dont la relation de réécriture est terminante et confluente, il existe donc une unique forme normale pour tout terme $t \in \mathcal{T}(\mathcal{F})$. On note cette dernière $t\downarrow_{\mathcal{R}}$.

Dans le cas d'un CBTRS, on s'intéresse souvent à des systèmes complets :

Définition 1.25 (Complétude)**.** *Étant donné un CBTRS* $\mathcal{R}$*, on dit que* $\mathcal{R}$ *est complet si et seulement si, pour tout symbole défini* $f \in \mathcal{D}^n$ *et pour toutes valeurs* $(v_1, \ldots, v_n) \in \mathcal{T}_{s_1}(\mathcal{C}) * \cdots * \mathcal{T}_{s_n}(\mathcal{C})$*, avec* $\mathcal{D}om(f) = s_1 * \cdots * s_n$*, il existe une règle* $f(l_1, \ldots, l_n) \twoheadrightarrow r \in \mathcal{R}$ *et une substitution* σ *telles* $v_i = \sigma(l_i)$*, pour tout* $i \in [1, n]$.

Étant donné un CBTRS complet $\mathcal{R}$, $t \in \mathcal{T}(\mathcal{F})$ est non-réductible dans le système $\mathcal{R}$ si et seulement si t est une valeur.

1.4 Stratégies

La relation de réécriture, comme définie dans la Section précédente, s'applique de manière exhaustive et non-déterministe. La notion de *Système abstrait de réduction* peut permettre de modéliser, pas à pas, toutes les transformations possibles d'un objet à un autre :

Définition 1.26 (Système abstrait de réduction [KKK08])**.** *Un système abstrait de réduction* (ARS : Abstract Reduction System) *est un graphe orienté étiqueté* $(\mathcal{O}, \mathcal{S})$*. Les nœuds* $\mathcal{O}$ *sont appelés objets, les arêtes orientées* $\mathcal{S}$ *sont appelées pas.*

Dans une algèbre de termes $\mathcal{T}(\mathcal{F},\mathcal{X})$, l'ARS correspondant à un système de réécriture $\mathcal{R}$ est donc représenté par le graphe $(\mathcal{T}(\mathcal{F},\mathcal{X}), \mathcal{S})$, où les arêtes correspondent à des pas de la relation de réécriture induite par $\mathcal{R}$ et sont étiquetées par le nom de la règle appliquée (*i.e.* étant données $u, v \in \mathcal{T}(\mathcal{F},\mathcal{X})$, il existe une arête étiquetée $\phi \in \mathcal{R}$ entre u et v si et seulement la règle ϕ réécrit u en v).

Définition 1.27 (Dérivation)**.** *Soit un ARS* $\mathcal{A}$ *:*

- *un* pas de réduction *est une arête étiquetée* ϕ *complétée de sa source* a *et de sa destination* b*. On note un pas de réduction* $a \Longrightarrow^{\phi}_{\mathcal{A}} b$*, ou simplement* $a \Longrightarrow^{\phi} b$ *lorsqu'il n'y a pas d'ambiguïté ;*
- *une* $\mathcal{A}$-dérivation *est un chemin* π *dans le graphe* $\mathcal{A}$ *;*
- *lorsque cette dérivation est finie,* π *peut s'écrire* $a_0 \Longrightarrow^{\phi_0} a_1 \Longrightarrow^{\phi_1} a_2 \cdots \Longrightarrow^{\phi_n} a_n$ *et on dit que* a_0 *se réduit en* a_n *par la dérivation* $\pi = \phi_0\phi_1\cdots\phi_{n-1}$ *; notée aussi* $a_0 \rightarrow^{\phi_0\phi_1\ldots\phi_{n-1}} a_n$ *ou simplement* $a_0 \rightarrow^{\pi} a_n$.
  - n *est la* longueur *de* π *;*
  - *la source de* π *est le singleton* $\{a_0\}$*, notée* $\mathcal{D}om(\pi)$ *;*
  - *la destination de* π *est le singleton* $\{a_n\}$*, notée* $\pi[a_0]$.

- *une dérivation est vide si elle n'est formée d'aucun pas de réduction. La dérivation vide de source a est notée id_a.*

Pour contrôler l'application d'une relation de réécriture, le concept de *stratégie* permet de sélectionner les dérivations qui permettront de garantir les propriétés voulues de la relation de réécriture, comme la terminaison et la confluence :

Définition 1.28 (Stratégie abstraite)**.** *Soit un ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$:*

- *une* stratégie abstraite *est un sous-ensemble de toutes les dérivations de $\mathcal{A}$;*
- *l'application de la stratégie ζ sur un objet a, noté par $\zeta[a]$, est l'ensemble de tous les objets atteignables depuis a en utilisant une dérivation dans ζ : $\zeta[a] = \{\pi[a] \mid \pi \in \zeta\}$. Lorsqu'aucune dérivation dans ζ n'a pour source a, on dit que l'application sur a de la stratégie a échoué ;*
- *appliquer la stratégie ζ sur un ensemble d'objets consiste à appliquer ζ à chaque élément a de l'ensemble. Le résultat est l'union de $\zeta[a]$ pour tous les a de l'ensemble d'objets ;*
- *la stratégie qui contient toutes les dérivations vides est $Id = \{id_a \mid a \in \mathcal{O}\}$.*

Pour expliciter le comportement de stratégie d'évaluation en *Appel par valeur*, on peut, par exemple, utiliser la stratégie de réduction stricte : l'ensemble des dérivations de $(\mathcal{T}(\mathcal{F}, \mathcal{X}), \mathcal{S})$ tels que pour chaque pas de réduction $a \Longrightarrow^{\phi} b$ avec $\phi : l \twoheadrightarrow r$, il existe une position $\omega \in \mathcal{P}os(a)$ et une substitution valeur ς telles que $a_{|\omega} = \varsigma(l)$ et $b = a\,[\varsigma(r)]_\omega$.

On définit directement la relation de réécriture sous stratégie de réduction stricte :

Définition 1.29 (Réécriture stricte)**.** *Étant donné un système de réécriture $\mathcal{R}$, on définit la relation de réécriture stricte sur $\mathcal{T}(\mathcal{F})$ telle qu'un terme $u \in \mathcal{T}(\mathcal{F})$ se réécrit strictement en $v \in \mathcal{T}(\mathcal{F})$ dans le système $\mathcal{R}$, notée $u \longrightarrow_{\mathcal{R}} v$, si et seulement s'il existe une règle $l \twoheadrightarrow r \in \mathcal{R}$, une position $\omega \in \mathcal{P}os(u)$ et une substitution valeur ς telles que :*

- $u_{|\omega} = \varsigma(l)$
- $v = u\,[\varsigma(r)]_\omega$

On note $\longrightarrow^*_{\mathcal{R}}$ la fermeture reflexive et transitive de $\longrightarrow_{\mathcal{R}}$.

En pratique, de nombreux langages à base de réécriture proposent des *langages de stratégie* permettant d'exprimer de manière déclarative certaines de ces stratégies. Nous présenterons certains de ces langages dans le Chapitre suivant.

2

Contexte et Motivations

L'intégration des technologies informatiques dans les systèmes critiques, en particulier dans le domaine de l'aérospatial et plus généralement du transport de personnes, a incité le développement de techniques et d'outils permettant de garantir la fiabilité et la sureté de ces technologies. Cependant, dans le contexte de la vérification d'un logiciel, il est non seulement très difficile de garantir l'absence d'erreur dans un programme informatique, mais une telle garantie est d'autant plus compliquée à fournir que le programme, lui-même, est compliqué. Dans le cas de systèmes critiques, les conséquences d'une erreur sont non seulement très couteuses, mais peuvent être désastreuses au point d'entrainer des pertes en vies humaines. Étant donnée la complexité des programmes considérés et l'omniprésence croissante des technologies informatiques, le développement de techniques permettant leur validation est ainsi devenu crucial.

Dans de nombreux cas, plutôt que d'essayer de fournir une preuve formelle de l'exactitude d'un logiciel, les ingénieurs et analystes en charge de sa conception se reposent sur des techniques tels que le test et la simulation pour valider leur produit. Ces techniques ont démontré leur utilité, puisqu'elles permettent très souvent d'identifier et de corriger de très nombreux *bugs*. Cependant elles ne permettent pas de démontrer formellement l'absence complète d'erreurs. De nombreuses techniques, basées sur le concept d'analyse statique, ont été proposées pour fournir une approche formelle permettant de prouver, de façon aussi exhaustive et définitive que possible, la correction d'un programme.

On détaillera dans la Section 2.3 un certain nombre de ces techniques, et notamment celles ayant pu influencer la méthode proposée dans cette thèse. On propose dans un premier temps d'introduire la problématique ayant inspiré cette méthode : la conception et la vérification de passes de compilation basées sur l'approche de Nanopass. La solution proposée étant principalement basée sur des travaux antérieurs dans le domaine de la réécriture, on considèrera également les différentes approches proposées par les langages de réécriture.

2.1 Problématique

La mise en place d'outils d'analyse statique pour la vérification formelle est généralement basée, d'une part, sur la conception et l'étude d'un langage dédié de spécification, et sur l'analyse syntaxique et sémantique des programmes, d'autre part. La compilation et la réécriture sont deux approches très courantes, basées sur l'étude syntaxique et sémantique de programmes, ainsi que sur des techniques d'analyse et de transformation de ces derniers. En s'inspirant de travaux menés dans le domaine de la compilation, la problématique considérée ici propose donc de s'intéresser à la structuration des compilateurs en passes de compilation pour proposer une approche

originale, basée sur des techniques de réécriture, permettant l'analyse statique de transformations de programmes.

2.1.1 Passes de compilation

Dans le domaine de la compilation, l'approche conventionnelle adoptée par une grande majorité de compilateurs consiste à structurer la transformation du code source en un programme exécutable via une série d'étapes, appelées passes de compilation. Chaque passe effectue une transformation entre différentes représentations intermédiaires du code compilé. Là où l'approche d'origine consistait à minimiser le nombre de passes pour optimiser le fonctionnement du compilateur, les systèmes modernes disposant d'une mémoire et d'une capacité de calcul bien plus grandes, l'augmentation du nombre de passes dans les compilateurs modernes a permis le développement de compilateurs capables de générer et d'optimiser le code de façon adaptée à de multiples architectures et systèmes d'exploitation.

Cette multiplication du nombre de passes dans les compilateurs a également mené à une simplification de la fonctionnalité de chaque passe. À partir de la description des langages intermédiaires d'entrée et de sortie, de telles passes de compilation réalisent ainsi généralement des transformations n'affectant qu'un petit nombre de constructions du langage d'origine. Par exemple, des passes effectuant des transformations de construction de sucre syntaxique vont éliminer les symboles correspondant au sucre syntaxique considéré du langage d'entrée, alors que des passes d'optimisation modifieront des constructions plus complexes.

Cette simplification des passes de compilation permet de gérer la complexité des compilateurs modernes en décomposant les fonctionnalités via des passes relativement simples. On peut ainsi concevoir des compilateurs de plus en plus complexes mais plus modulaires, ce qui facilite le travail de relecture, *debugging* et l'ajout de fonctionnalités. De plus, en détaillant les langages d'entrée et de sortie de chaque passe, il est possible de spécifier des garanties syntaxiques de la transformation et de vérifier la transformation effectuée par la passe par rapport à ces spécifications.

Considérons un langage de λ-expressions dont les constructions de base sont les λ-termes, l'application, les définitions *let* et les variables (identifiées par une chaîne de caractère). Algébriquement on peut représenter ce langage par le type algébrique suivant (où String est considéré comme le type intégré de chaînes de caractères) :

$$\begin{array}{rcl} \mathsf{Expr} & := & \mathit{lambda}(\mathsf{String}, \mathsf{Expr}) \\ & | & \mathit{apply}(\mathsf{Expr}, \mathsf{Expr}) \\ & | & \mathit{let}(\mathsf{String}, \mathsf{Expr}, \mathsf{Expr}) \\ & | & \mathit{var}(\mathsf{String}) \end{array}$$

Dans ce langage, la construction *let* peut être vue comme un sucre syntaxique : un terme de la forme $\mathit{let}(s, e_1, e_2)$ est équivalent au terme $\mathit{apply}(\mathit{lambda}(s, e_2), e_1)$. On peut donc concevoir une passe de compilation destiné à éliminer ce sucre syntaxique comme explicité. Une telle passe effectue une transformation depuis le langage précédent vers le langage cible suivant :

$$\begin{array}{rcl} \mathsf{ExprNoLet} & := & \mathit{lambda}(\mathsf{String}, \mathsf{Expr}) \\ & | & \mathit{apply}(\mathsf{Expr}, \mathsf{Expr}) \\ & | & \mathit{var}(\mathsf{String}) \end{array}$$

En spécifiant ainsi le langage de départ et le langage cible, on fournit une propriété syntaxique détaillant le fonctionnement attendu de la passe considérée. Il est alors possible d'utiliser le

système de types du langage d'implémentation pour vérifier que les types d'entrée et de sortie de la transformation correspondent bien à des expressions du langage de départ et du langage cible, respectivement.

2.1.2 Nanopass

La conception d'un compilateur utilisant ce modèle de passes est cependant assez couteuse et chronophage. En effet, dans l'exemple considéré, les langages d'entrée et de sortie n'ont qu'un faible nombre de constructions, mais en pratique, ce type de passe doit être implémenté sur des langages pouvant contenir plusieurs dizaines de constructions différentes. Déclarer les langages intermédiaires de chaque passe et le détail des mécanismes d'une passe n'affectant qu'un petit nombre de constructions devient alors un travail fastidieux, et très souvent redondant.

La librairie de compilation Nanopass [Kee13] propose de simplifier l'écriture de ce type de passes de compilation en fournissant des constructions permettant de gérer les redondances liées à leur développement et de générer une grande partie du code définissant les différents langages intermédiaires et les transformations associées. En particulier, la commande `define-language` permet de définir un langage, soit explicitement en déclarant l'ensemble des constructions du langage, soit par comparaison à un langage d'origine, en déclarant les constructions à ajouter et à enlever. Tandis que la commande `define-pass` permet de définir la transformation effectuée par une passe en explicitant, d'une part, les langages d'entrée et de sortie de la passe, et d'autre part, les opérations effectuées par la transformation elle-même.

En reprenant l'exemple présenté dans la section précédente, on peut ainsi déclarer le langage initial d'expressions avec un `define-language` explicite :

```
(define-language Lsource
  (terminals
    (variable (x)))
  (Expr (e e1 e2)
    x
    (lambda x e)
    (apply e1 e2)
    (let x e1 e2)))
```

Dans ce langage `Lsource`, les variables sont considérées comme des constructions terminales (les seules de notre langage) identifiées par la méta-variable `x`. En pratique, il faudrait également écrire un prédicat vérifiant la forme d'une telle variable. On définit ensuite le type Expr, identifié par les méta-variables `e`, `e1` et `e2`, étant une variable ou une construction non-terminale `lambda`, `apply` ou `let`.

On peut ensuite définir le langage d'expressions cible obtenu après l'élimination de la construction *let*, par extension du langage `Lsource` :

```
(define-language Lnolet
  (extends Lsource)
  (Expr (e e1 e2)
    (- (let x e1 e2))))
```

Dans ce cas, on ne décrit que les différences avec le langage étendu, *i.e.* les constructions (terminales ou non) à ajouter ou à retirer du langage. Ici, on veut juste retirer la construction `let` comme explicité par la clause `(- (let x e1 e2))` [1]. Cette définition du langage `Lnolet` est

1. on peut également ajouter des constructions avec une clause `(+ (if e e1 e2))`, par exemple.

donc équivalente à la définition explicite suivante :

```
(define-language Lnolet
  (terminals
    (variable (x)))
  (Expr (e e1 e2)
    x
    (lambda x e)
    (apply e1 e2)))
```

Cette approche permet de réduire les redondances dans les définitions des langages intermédiaires en ne déclarant que les modifications entre chaque langage. Ainsi plutôt, que de devoir re-déclarer l'ensemble des constructions de chaque langage, on se contente de spécifier le faible nombre de constructions affectées par la passe de compilation et on se repose sur la librairie Nanopass pour générer la définition concrète explicite de chaque langage.

Une fois les langages d'entrée et de sortie d'une passe correctement déclarés, on peut en spécifier le fonctionnement avec la commande `define-pass` :

```
(define-pass remove-let : Lsource (x) -> Lnolet ()
  (Expr : Expr (ir) -> Expr ()
    [,x x]
    [(lambda ,x ,e) `(lambda ,x ,(Expr e))]
    [(apply ,e1 ,e2) `(apply ,(Expr e1) ,(Expr e2))]
    [(let ,x ,e1 ,e2) `(apply (lambda ,x ,(Expr e2)) ,(Expr e1))]))
```

On définit ici une passe transformant un programme du langage d'entrée `Lsource` vers le langage cible `Lnolet`. Dans cette passe, on définit la transformation, nommée `Expr`, des non-terminaux `Expr` de `Lsource` vers les non-terminaux `Expr` de `Lnolet`. Pour chaque construction d'`Expr`, on définit le comportement de cette transformation avec le motif correspondant en entrée et une expression en sortie. On remarque qu'il faut ici expliciter l'appel récursif à `Expr` pour effectuer la transformation des termes de `Lsource Expr` vers des termes de `Lnolet Expr`. En pratique, la construction `define-pass` déclare l'application d'un tel catamorphisme [MFP91], en mettant les variables sur lesquels se fait l'appel récursif entre crochet :

```
(define-pass remove-let : Lsource (x) -> Lnolet ()
  (Expr : Expr (ir) -> Expr ()
    [,x x]
    [(lambda ,x ,[e]) `(lambda ,x ,e)]
    [(apply ,[e1] ,[e2]) `(apply ,e1 ,e2)]
    [(let ,x ,[e1] ,[e2]) `(apply (lambda ,x ,e2) ,e1)]))
```

Les méta-variables `e`, `e1` et `e2` dans les motifs de chaque clause de la transformation `Expr` deviennent ainsi `[e]`, `[e1]` et `[e2]` pour indiquer l'application du catamorphisme. Cette notation simplifie déjà l'écriture de la passe, mais on voit d'autant plus clairement que la plupart des clauses de la transformation `Expr` se contente de reconstruire le terme du langage d'origine `Lsource` dans le langage cible `Lnolet` (ici, c'est le cas des clauses de variable, *lambda* et *apply*). Étant données les définitions de ces deux langages, la commande `define-pass` est capable de générer automatiquement ces clauses. On peut donc simplement définir la passe en n'explicitant que les clauses modifiant concrètement une construction du langage d'entrée :

```
(define-pass remove-let : Lsource (x) -> Lnolet ()
  (Expr : Expr (ir) -> Expr ()
```

```
[(let ,x ,[e1] ,[e2]) `(apply (lambda ,x ,e2) ,e1)]))
```

L'approche proposée par Nanopass permet ainsi de réduire nettement l'effort nécessaire à l'écriture de ce type de passe de compilation, tout en proposant un système de type permettant de vérifier syntaxiquement la transformation effectuée. La librairie a été utilisée aussi bien dans un contexte éducatif [SWD04], que commercial [KD13] où elle a été utilisée pour développer un compilateur du langage Scheme. Cette approche a également été implémentée dans une librairie OCaml [Mil17]. Du point de vue des domaines de l'analyse statique et de la vérification formelle, l'approche proposée par Nanopass reste un peu superficielle puisqu'elle ne permet que de vérifier l'absence ou la présence de symboles dans le résultat d'une transformation. On propose cependant de s'en inspirer pour présenter une approche algébrique basée sur la notion de filtrage par motif.

2.1.3 Approche algébrique

L'approche proposée par Nanopass repose sur le système de type sous-jacent du langage d'implémentation pour vérifier qu'une passe de compilation effectue bien une transformation depuis un langage de départ vers un langage cible. Ce type d'approche est néanmoins limité par l'expressivité de représentation de ces deux langages. Dans Nanopass, cette expressivité est suffisante pour exprimer des modifications qui ne consistent qu'en l'ajout ou la soustraction de quelques symboles du langage.

Dans un contexte plus général, on voudrait pouvoir exprimer que les termes obtenus via une transformation vérifient des propriétés plus complexes pouvant considérer et combiner de multiples symboles. Donner une telle caractérisation de l'ensemble des termes obtenus par transformation est en effet une approche utilisée par de nombreux outils d'analyse statique. Par exemple, des approches tels que la complétion d'automate [Gen14] ou le *model checking* [MB04], proposent de modéliser un programme sous la forme d'un système de réécriture ou d'un transducteur et de vérifier la correction du programme en déterminant la forme syntaxique des termes obtenus.

En reprenant l'exemple du langage de λ-expressions présenté dans les Sous-Sections précédentes, on pourrait considérer la définition d'un système de type permettant de vérifier que les expressions sont en *continuation-passing style* (CPS) [Rey93] ou en *A-normal form* (ANF) [SF92]. Ces formes d'expressions sont particulièrement utiles pour la compilation et l'optimisation de langages fonctionnels puisqu'elles permettent, notamment, de garantir que toutes les récursions sont terminales. Pour garantir que les expressions considérées sont bien de la forme voulue, on aurait alors besoin de déclarer un système de type spécifiquement conçu pour correspondre à la propriété souhaitée. Par exemple, pour des expressions en ANF, on peut définir les types suivant :

$$\begin{array}{rcl} \mathsf{Expr} & := & \mathit{val}(\mathsf{Val}) \\ & | & \mathit{let}(\mathsf{String}, \mathsf{Val}, \mathsf{Expr}) \\ & | & \mathit{letapply}(\mathsf{String}, \mathsf{Val}, \mathsf{Val}, \mathsf{Expr}) \end{array} \qquad \begin{array}{rcl} \mathsf{Val} & := & \mathit{lambda}(\mathsf{String}, \mathsf{Expr}) \\ & | & \mathit{var}(\mathsf{String}) \end{array}$$

Bien que certaines approches fonctionnelles, telles que les variants polymorphiques [Gar98], puissent simplifier la définition de systèmes de type, combinant des notions de sous-typage, polymorphisme et surcharge, spécifiquement conçus pour vérifier la propriété de correction souhaitée, il reste néanmoins nécessaire de construire ces systèmes au cas par cas. On propose donc plutôt de se reposer sur une approche algébrique où les propriétés recherchées seront exprimées par des notions de filtrage par motif.

Plus précisément, on va chercher à garantir la correction d'une transformation en vérifiant qu'un motif donné est absent du résultat de la transformation considérée. Pour reprendre l'approche de Nanopass, cela revient à vérifier que les termes du langage de sortie de la passe ne

contiennent aucun sous-terme filtré par un motif dont le symbole de tête est un symbole éliminé du langage cible. Dans le cas de termes en CPS ou ANF, on peut, par exemple, vérifier que toutes les applications sont terminales : *i.e.* les termes ne contiennent pas d'application à gauche d'une application.

Cette approche permet de vérifier de nombreuses propriétés allant de l'aplatissement ou du tri d'une liste à la forme de formules logiques (on présente en Annexe C, un certain nombre d'exemples qui ont été validés par l'implémentation de la méthode présentée, et dont certains seront détaillés dans les chapitres suivants). On peut également mentionner la notion de sécurité de filtrage dans les langages fonctionnels, qui a fait l'objet de travaux [JR21] cherchant à proposer des méthodes pour vérifier qu'un programme fonctionnel ne peut pas échouer à cause d'une fonction définie par filtrage de manière non-exhaustive.

2.2 Langages à base de réécriture

La réécriture est un formalisme très répandu dans de nombreux domaines de l'informatique, en particulier dans les domaines de la vérification formelle et de la transformation de langage. En effet, de par son aspect formel et son expressivité, c'est un formalisme particulièrement bien adapté pour décrire des sémantiques de langages.

Dans le contexte des langages fonctionnels, le formalisme de réécriture fournit un cadre idéal pour illustrer les programmes considérés en les traduisant sous la forme de CBTRSs. En effet, comme on l'a vu dans le Chapitre 1, les termes considérés par les programmes fonctionnels peuvent être décrits sous la forme d'une algèbre de termes sortée, en traduisant les types algébriques sous la forme d'une signature sortée. Ainsi, on peut considérer un programme fonctionnel comme manipulant des termes de l'algèbre ainsi obtenue et le traduire sous la forme d'un système de réécriture :

```
removeLet :: Expr -> Expr
removeLet (Lambda x e) = Lambda x (removeLet e)
removeLet (Apply e1 e2) = Apply (removeLet e1)
                                (removeLet e2)
removeLet (Let x e1 e2) = Apply (Lambda x (removeLet e2))
                                (removeLet e1)
removeLet (Var x) = Var x
```

$$
\begin{cases}
removeLet(lambda(x, e)) & \rightarrow & lambda(x, removeLet(e)) \\
removeLet(apply(e_1, e_2)) & \rightarrow & apply(removeLet(e_2), \\
 & & \quad removeLet(e_1)) \\
removeLet(let(x, e_1, e_2)) & \rightarrow & apply(lambda(x, removeLet(e_2)), \\
 & & \quad removeLet(e_2)) \\
removeLet(var(x)) & \rightarrow & var(x)
\end{cases}
$$

FIGURE 2.1 – Traduction de la fonction *removeLet* en Haskell (en haut) sous la forme d'un CBTRS (en bas)

De nombreux langages ont ainsi proposé d'implémenter des notions similaires de filtrage par motif et de règles de réécriture. On présente dans cette section un tour d'horizon de ces langages à base de réécriture.

2.2.1 Réécriture par stratégies

Une des premières problématiques qui a été considérée par les langages à base de réécriture a été de contrôler l'application des règles. Pour pouvoir gérer les problématiques de confluence et de terminaison, de nombreux langage ont en effet proposé de limiter l'application des règles de réécriture via des combinateurs de stratégie particulièrement adaptés à la description de parcours d'arbres.

ELAN

Le langage ELAN [BKK+98] est un des premiers langages à introduire des constructions de stratégie permettant un tel contrôle. Ce développement se présente dans un premier temps par la distinction entre deux sortes de règles : les règles anonymes s'appliquant systématiquement (suivant une stratégie *innermost* : en profondeur d'abord) et les règles étiquetées pouvant ainsi être appelées et contrôlées via une stratégie spécifique. Une autre particularité d'ELAN est sa gestion du non-déterminisme, puisque l'application d'une règle étiquetée renvoie un ensemble de termes obtenus suivant la stratégie choisie.

Pour définir les stratégies utilisées, ELAN introduit un langage de stratégies [KKV93] proposant un ensemble de combinateurs de stratégies permettant de les composer et de les contrôler. Une stratégie est ainsi considérée comme un objet à part entière du langage. Les règles étiquetées, l'identité et l'échec forment un ensemble de stratégies élémentaires, s'appliquant directement à la racine du terme considéré, et dont l'application est contrôlée plus finement via ces combinateurs. Ces derniers permettent à la fois de détailler l'ordre d'application des règles et de contrôler cette application en fonction du contexte. Pour encoder la stratégie de traversée d'arbre, ELAN génère également, pour chaque symbole f, un opérateur de congruence F de même arité permettant d'expliciter les stratégies appliquées à chaque sous-terme d'un terme ayant f comme symbole de tête.

Par la suite des opérateurs de traversée polymorphes ont été introduits, via des fonctions des destructions/constructions polymorphiques, permettant ainsi de définir des stratégies de traversée indépendamment de la signature choisie. En permettant également de définir des stratégies de manière récursive, ELAN propose ainsi un langage de stratégie qui a inspiré, par la suite, de multiples langages.

Stratego

Un des principaux langages à hériter des principes de stratégies introduits par ELAN est le langage de transformation de programmes Stratego [VBT98]. Il développe énormément le concept de construction de stratégie en privilégiant des combinateurs de stratégie polymorphiques. En plus des combinateurs de séquence, de choix (déterministe et non-déterministe), le langage de stratégie proposé par Stratego est, grâce à cette approche polymorphique, le premier à intégrer des combinateurs de congruence génériques :

- l'opérateur `i`(s) permet d'appliquer la stratégie s au $i^{\text{ème}}$ sous-terme du terme, et échoue si l'arité du symbole de tête est trop petite ;
- l'opérateur `All`(s) applique s à tous les sous-termes du terme considéré, et échoue si l'application de s échoue pour au moins un des sous-termes ;
- l'opérateur `One`(s) applique s au premier sous-terme pour lequel s n'échoue pas, et échoue si l'application de s échoue pour tous les sous-termes ;
- l'opérateur `Some`(s) applique s à tous les sous-termes pour lesquels s n'échoue pas, et échoue si l'application de s échoue pour tous les sous-termes.

Pour gérer la définition de stratégie récursive, Stratego introduit également l'opérateur $\mu\mathtt{x}(s)$, où `x` peut être utilisé dans la construction de s pour renvoyer à la définition récursive.

Grâce à l'expressivité de ce langage de stratégie, Stratego a démontré être un langage particulièrement bien adapté à la transformation de langage [Vis01]. En effet, de nombreuses stratégies, utilisées pour les transformations d'arbres syntaxiques, telles que *top-down* (du haut vers les bas), *bottom-up* (du bas vers le haut) et *innermost* (en profondeur d'abord), sont facilement encodables via les différents combinateurs de stratégie proposés.

Maude

Le langage de spécification Maude [CELM96] est basé, comme les deux précédents, sur la réécriture, mais propose originellement une approche fondamentalement différente au contrôle de l'application des règles. En effet, les objets du langage Maude s'appuient sur une représentation à un *méta-niveau*. Il est possible d'appliquer des règles sur cette représentation méta via l'opérateur `meta-apply`. Cet opérateur normalise le terme et retourne la méta-représentation du terme résultant de l'évaluation.

L'application d'un ensemble de règles peut donc être contrôlée en s'appuyant sur les capacités réflexives du langage : on contrôle l'application des règles de réécriture via un autre programme défini par réécriture. Cette approche, bien que très expressive, est particulièrement compliquée à utiliser d'un point de vue pratique. Par la suite [MMV04], Maude a ainsi intégré un langage de stratégie inspiré d'ELAN et Stratego.

Tom

Le langage Tom [BBK+07] est conçu sur le concept des *îlots formels* [Spi01, BKM06] pour enrichir des langages généralistes populaires, comme Java. Ces *îlots formels* permettent d'intégrer dans ces langages hôtes des concepts tels que le filtrage par motif et d'autres fonctionnalités issues de la réécriture, en restant aussi peu intrusif que possible. Pour cela, les constructions de Tom sont interprétées, transformées et compilées dans le langage hôte.

Parmi les fonctionnalités intégrées dans Tom, on retrouve des constructions permettant de définir des signatures algébriques via le module gom qui se repose sur le typage statique fort de Java pour générer un système de type équivalent à la signature multi-sortée considérée. Il est ensuite possible de définir des méthodes agissant sur les constructions algébriques ainsi générées via des modules de filtrage associatif et de stratégie.

Le langage Tom intègre, en effet, un langage de stratégie inspiré d'ELAN et Stratego. Comme dans ces derniers, l'identité, l'échec et les règles de réécriture sont toujours les constructions de stratégie élémentaire qui sont ensuite composées via un ensemble de combinateurs similaire. À la différence de ces langages, une règle de réécriture est définie comme stratégie en étendant une stratégie par défaut (généralement l'identité ou l'échec), ce qui permet d'expliciter le comportement par défaut de la stratégie.

2.2.2 Conception et Analyse par Réécriture

De nombreux autres outils se sont inspirés du formalisme de réécriture pour concevoir des environnements destinés à la conception de langages, notamment pour les langages dédiés (*DSL : Domain Specific Language*), et/ou l'analyse de programmes. La réécriture est en effet un formalisme particulièrement adapté à la définition de transformations d'arbres syntaxiques et à la description de la sémantique de programmes.

ASF+SDF

Le formalisme ASF+SDF [vdBvDH+01] est une combinaison construite autour des langages ASF (*Algebraic Specification Formalism*) et SDF (*Syntax Definition Formalism*). Ces deux formalismes permettent de définir un langage en décrivant à la fois les aspects syntaxiques du langage et en en fournissant une spécification sous la forme d'équations algébriques.

Un des intérêts de la combinaison ASF+SDF vient du fait qu'il est possible d'utiliser la syntaxe décrite dans SDF directement dans les spécifications ASF. Ces dernières étant écrites sous la forme d'équations algébriques, elles peuvent être interprétées comme des règles de réécriture et appliquées de façon *innermost* (en profondeur d'abord), ou via une fonction de traversée définie par l'utilisateur.

Un environnement de développement a été construit autour du formalisme ASF+SDF en l'intégrant avec une interface utilisateur, des éditeurs, parsers, compilateurs et interpréteurs.

Rascal

Héritier direct du formalisme ASF+SDF, Rascal [KvdSV09] offre une interface complète destinée à la transformation et l'analyse de programmes. Autour des syntaxe et sémantique de ASF+SDF, Rascal s'est inspiré de nombreux outils ayant influencé l'évolution des techniques de transformation et d'analyse de code source, et a notamment intégré de nombreuses fonctionnalités construites autour de la logique de réécriture et du filtrage par motif.

Un point central de la philosophie du langage Rascal est sa volonté de s'intégrer avec des outils de l'état de l'art pour proposer des fonctionnalités supplémentaires et destinées à un plus grand public. Par exemple, il est possible de déléguer le processus d'analyse à Maude ou K [HKV12] : pour cela, le code Rascal est transformé pour correspondre à la syntaxe du langage utilisé, la tâche d'analyse est générée et exécutée. Une fois le résultat de l'analyse obtenu, ce dernier est interprété par Rascal et remi dans le contexte du code analysé.

De plus, Rascal est également intégré sous la forme d'un plugin Eclipse permettant l'analyse et la transformation de code source directement au sein de l'environnement de développement.

Spoofax

De façon similaire à Rascal, Spoofax [KV10, VWT+14] propose une plateforme destinée à la conception et l'analyse de langage dédié en intégrant le formalisme `SDF` avec le langage de réécriture stratégique Stratego. Cette plateforme permet non seulement de considérer les aspects syntaxiques et sémantiques des langages mais aussi de fournir un ensemble de fonctionnalités allant du parsing, à la génération de code exécutable en passant par des outils de vérification de spécifications sémantiques, et de transformations.

Au-delà des aspects syntaxiques et sémantiques, la plateforme Spoofax permet la définition de règles de résolution de nommage en fonction de la portée, ainsi que des règles d'analyse et d'inférence de type. A partir de l'ensemble des spécifications du langage dédié, Spoofax génère alors parsers, outils d'analyse et de vérification, et interpréteurs pour le langage ainsi défini.

Tout comme Rascal, Spoofax est construit autour d'une intégration à l'environnement de développement Eclipse, et s'intègre à d'autres outils, comme l'assistant de preuve formelle Coq.

L'approche algébrique proposée par le formalisme de réécriture le rend particulièrement bien adapté à l'écriture de transformation d'arbre syntaxiques comme proposé dans la Section précédente. Des mécanismes, tels que les stratégies de réécriture, utilisés par certains langages peuvent même grandement faciliter l'écriture de ce type de transformation. Mais bien qu'il y ait une forte interdépendance entre la notion de filtrage par motif et la relation de réécriture, la plupart des

langages à base de réécriture sont destinés à l'analyse de code ou la conception de langage dédié et n'apportent que très peu de garanties sur les transformations effectuées.

2.2.3 Génération de règle et Filtrage

On voit par le nombre d'outils qui se reposent sur des notions de réécriture, que ce formalisme est particulièrement bien adapté à l'écriture de transformations. Mais de par son expressivité et son aspect algébrique, c'est également un formalisme qui se porte tout aussi bien à l'étude formelle de ces transformations. L'utilisation de systèmes de réécriture permet, par exemple, de considérer des notions de terminaison, qui peuvent être vérifiées par des outils tels que AProVE [FGP$^+$11] ou TTT2 [KSZM09], ou bien encore de confluence.

Aussi, dans des approches de réécriture stratégique telles que celles proposées dans les langages Stratego ou Tom, il peut être intéressant de se ramener à un système de réécriture classique pour pouvoir en étudier les propriétés. C'est ce que propose [CLM15] : pour chaque règle de réécriture et chaque combinateur de stratégie utilisé, il est possible de générer des symboles et des règles de réécriture associées permettant d'encoder la stratégie construite dans un système de réécriture classique. Cette approche, prouvée correcte et complète, permet non seulement l'étude des propriétés de terminaison des stratégies, mais peut également servir d'encodage d'un langage de stratégie dans tout langage supportant les concepts de réécriture (comme ceux présentés précédemment), voir simplement des notions de filtrage par motif (comme la plupart des langages fonctionnels généralistes).

De façon similaire, les langages fonctionnels, ou certains langages à base de réécriture, adoptent en général un formalisme de système de réécriture ordonné : les règles sont appliquées dans l'ordre de préférence dans lequel elles sont spécifiées, *i.e.* si deux règles peuvent s'appliquer, la première des deux s'appliquera toujours exclusivement. Ce type d'approche permet généralement une définition plus concise et claire d'un système de réécriture. Comme dans le cas des stratégies, les travaux présentés dans [CM19] proposent d'encoder ce type de système de réécriture ordonné en un TRS classique.

L'approche proposée dans [CM19] passe par l'étude de motifs étendus linéaires, comme définis dans la Section 1.2, et de la notion de filtrage par motif associée. Pour cela, une notion de sémantique close de motif est introduite :

Définition 2.1 (Sémantique close)**.** *Étant donné un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on définit la sémantique close de t, notée $[\![t]\!]$, comme l'ensemble des instances de t : $[\![t]\!] = \{\sigma(t) \mid \forall \sigma.\sigma(t) \in \mathcal{T}(\mathcal{C})\}$.*

Pour un motif étendu linéaire, on définit la sémantique close de manière récursive :

- $[\![x_s]\!] = \mathcal{T}_s(\mathcal{C})$*, pour toute variable* $x_s \in \mathcal{X}$ *;*
- $[\![c(p_1, \ldots, p_n)]\!] = \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![p_1]\!] \times \cdots \times [\![p_n]\!]\}$ *pour tout constructeur* $c \in \mathcal{C}$ *et motifs* $p_1, \ldots, p_n$ *;*
- $[\![p_1 + p_2]\!] = [\![p_1]\!] \cup [\![p_2]\!]$ *pour tout motifs* p_1 *et* p_2 *;*
- $[\![p_1 \setminus p_2]\!] = [\![p_1]\!] \setminus [\![p_2]\!]$ *pour tout motifs* p_1 *et* p_2 *;*
- $[\![\bot]\!] = \emptyset$*.*

En partant de l'observation qu'un motif de l'algèbre peut être interprété comme l'ensemble de ses instances, cette notion de sémantique permet de représenter l'ensemble (potentiellement infini) des termes filtrés par un motif étendu par une structure finie, puisqu'on a :

Proposition 2.1. *Étant donné un motif étendu linéaire p et une valeur v, on a $p \preccurlyeq v$ si et seulement si $v \in [\![p]\!]$.*

Étant donné un système de réécriture ordonné :

$$\begin{cases} l_1 & \twoheadrightarrow & r_1 \\ l_2 & \twoheadrightarrow & r_2 \\ \vdots & & \vdots \\ l_n & \twoheadrightarrow & r_n \end{cases}$$

le but de l'approche proposée est de résoudre le problème de *désambiguïsation* [Kra08] des motifs $[l_1, \ldots, \mathrm{ln}]$, *i.e.* de déterminer, pour tout $i > 1$, un ensemble de motifs P_i tel que $\bigcup_{p \in P_i} [\![p]\!] = [\![l_i]\!] \setminus \bigcup_{j=1}^{i-1} [\![l_j]\!]$. On peut alors pour chaque règle $l_i \twoheadrightarrow r_i$ remplacer la règle dans le système par un ensemble de règles ayant pour membre gauche les motifs de P_i.

Exemple 2.1. *On considère l'algèbre définie par la signature $\Sigma_{let} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par le type algébrique :*

$$\begin{array}{rcl} \mathsf{Expr} & := & \mathit{lambda}(\mathsf{String}, \mathsf{Expr}) \\ & | & \mathit{apply}(\mathsf{Expr}, \mathsf{Expr}) \\ & | & \mathit{let}(\mathsf{String}, \mathsf{Expr}, \mathsf{Expr}) \\ & | & \mathit{var}(\mathsf{String}) \end{array}$$

et avec $\mathcal{D} = \{\mathit{removeLet}\}$. On étudie une version ordonnée du système de réécriture, présenté dans la Figure 2.1 , décrivant le comportement associé au symbole removeLet :

$$\begin{cases} \mathit{removeLet}(\mathit{lambda}(x, e)) & \twoheadrightarrow & \mathit{lambda}(x, \mathit{removeLet}(e)) \\ \mathit{removeLet}(\mathit{apply}(e_1, e_2)) & \twoheadrightarrow & \mathit{apply}(\mathit{removeLet}(e_2), \\ & & \qquad \mathit{removeLet}(e_1)) \\ \mathit{removeLet}(\mathit{let}(x, e_1, e_2)) & \twoheadrightarrow & \mathit{apply}(\mathit{lambda}(x, \mathit{removeLet}(e_2)), \\ & & \qquad \mathit{removeLet}(e_2)) \\ \mathit{removeLet}(x) & \twoheadrightarrow & x \end{cases}$$

Dans ce système de réécriture ordonné, la dernière règle ne peut s'appliquer que sur des termes pour lesquels toutes les autres règles ne s'appliquent pas. Donc le terme instanciant x ne peut pas être filtré par $\mathit{lambda}(x, e)$, $\mathit{apply}(e_1, e_2)$ ou $\mathit{let}(x, e_1, e_2)$. On cherche donc un ensemble de motifs P tel que $\bigcup_{p \in P} [\![p]\!] = [\![x_{\mathsf{Expr}}]\!] \setminus ([\![\mathit{lambda}(x, e)]\!] \cup [\![\mathit{apply}(e_1, e_2)]\!] \cup [\![\mathit{let}(x, e_1, e_2)]\!])$

Étant donnée la signature Σ_{let}, on a $P = \{\mathit{var}(s)\}$. On peut donc remplacer la dernière règle par $\mathit{removeLet}(\mathit{var}(s)) \twoheadrightarrow \mathit{var}(s)$, et on retrouve bien ainsi le système de la Figure 2.1.

Pour résoudre le problème de *désambiguïsation*, Cirstea et Moreau proposent le système de réécriture $\mathfrak{R}_\setminus$ permettant de réduire un motif étendu en un motif additif (*i.e.* ne contenant pas de complément $\setminus$) et préservant la sémantique.

Cirstea et Moreau montrent dans [CM19] que le système préserve la sémantique close des motifs :

Proposition 2.2. *Étant donnés des motifs linéaires u, v tels que $u \Longrightarrow_{\mathfrak{R}_\setminus} v$, on a $[\![u]\!] = [\![v]\!]$.*

Et ils garantissent l'existence et la structure des formes normales :

Proposition 2.3. *Le système $\mathfrak{R}_\setminus$ est confluent et terminant. Étant donné un motif étendu p, la forme normal $p \downarrow_{\mathfrak{R}_\setminus}$ est soit $\bot$, soit une somme de motifs constructeurs, i.e. un motif additif pure tel que pour toute position $\omega \in \mathcal{P}os(p \downarrow_{\mathfrak{R}_\setminus})$ si $p \downarrow_{\mathfrak{R}_\setminus} (\omega) = +$, alors $\forall \omega' < \omega, p \downarrow_{\mathfrak{R}_\setminus} (\omega') = +$.*

Élimine ensemble vide :

(A1) $\bot + \overline{v} \Rightarrow \overline{v}$

(A2) $\overline{v} + \bot \Rightarrow \overline{v}$

Distribution :

(E1) $\delta(\overline{v_1}, \ldots, \bot, \ldots, \overline{v_n}) \Rightarrow \bot$

(S1) $\delta(\overline{v_1}, \ldots, \overline{v_i} + \overline{w_i}, \ldots, \overline{v_n}) \Rightarrow \delta(\overline{v_1}, \ldots, \overline{v_i}, \ldots, \overline{v_n}) + \delta(\overline{v_1}, \ldots, \overline{w_i}, \ldots, \overline{v_n})$

Simplifie compléments :

(M1) $\overline{v} \setminus \overline{x}_s \Rightarrow \bot$

(M2) $\overline{v} \setminus \bot \Rightarrow \overline{v}$

(M3) $\overline{v} \setminus (\overline{w_1} + \overline{w_2}) \Rightarrow (\overline{v} \setminus \overline{w_1}) \setminus \overline{w_2}$

(M4) $\overline{x}_s \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n}) \Rightarrow \sum_{c \in \mathcal{C}_s} c(z_{1_{s_1}}, \ldots, z_{m_{s_m}}) \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n})$ *avec* $m = arity\,(c)$

(M5) $\bot \setminus \overline{v} \Rightarrow \bot$

(M6) $(\overline{v} + \overline{w}) \setminus \alpha(\overline{v_1}, \ldots, \overline{v_n}) \Rightarrow (\overline{v} \setminus \alpha(\overline{v_1}, \ldots, \overline{v_n})) + (\overline{w} \setminus \alpha(\overline{v_1}, \ldots, \overline{v_n}))$

(M7) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n}) \Rightarrow \alpha(\overline{v_1} \setminus \overline{t_1}, \ldots, \overline{v_n}) + \cdots + \alpha(\overline{v_1}, \ldots, \overline{v_n} \setminus \overline{t_n})$

(M8) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \beta(\overline{w_1}, \ldots, \overline{w_m}) \Rightarrow \alpha(\overline{v_1}, \ldots, \overline{v_n})$ *avec* $\alpha \neq \beta$

FIGURE 2.2 – $\mathfrak{R}_\setminus$: réduction de motifs étendus ; $\overline{u}, \overline{v}, \overline{v_1}, \ldots, \overline{v_n}$, $\overline{w}, \overline{w_1}, \ldots, \overline{w_n}$ filtre les motifs additifs, $\overline{t_1}, \ldots, \overline{t_n}$ filtre les motifs pures additifs, $\overline{x}$ filtre les variables. α, β s'étend à tous les symboles de $\mathcal{C}$, et δ de $\mathcal{C}^{n>0}$.

Pour tout motif étendu linéaire p, $t = p \downarrow_{\mathfrak{R}_\setminus}$ est donc un motif additif équivalent à p, *i.e.* pour toute valeur v, $p \not\ll v$ si et seulement si $t \not\ll v$.

La problématique considérée dans cette thèse consistant à vérifier qu'un motif est absent du résultat d'une transformation donnée, le formalisme proposé s'inspire de l'approche sémantique présentée dans [CM19]. En effet, grâce au système $\mathfrak{R}_\setminus$, on a vu que cette sémantique permet de représenter et de calculer des anti-motifs [KKM07], ou plus généralement des compléments de motifs, comme un ensemble de motifs constructeurs. On pourra donc étendre cette approche pour construire une sur-approximation de l'ensemble attendu des résultats d'une transformation et comparer cette sur-approximation aux termes obtenus par réduction suivant le TRS donné pour décrire la transformation considérée.

2.3 Analyse statique

Le concept d'analyse statique de programmes englobe l'ensemble des méthodes développées dans le but de vérifier et de décrire le comportement d'un programme, sans l'exécuter (contrairement aux méthodes dites "dynamiques", comme le test ou la simulation, qui requiert son exécution). De nombreux outils et méthodes d'analyse ont ainsi proposé de vérifier qu'un programme fournit une solution à un problème donné.

On propose dans cette section de présenter quelques approches différentes de la littérature qui ont pour objectif de fournir des garanties pouvant envelopper, ou au moins intersecter, la problématique introduite précédemment.

2.3.1 Langages réguliers

Afin de garantir la forme des termes obtenus par transformation suivant une relation de réécriture, une approche répandue consiste à représenter les termes issus de la transformation comme des termes d'un langage régulier.

Une des premières approches proposant une méthode pour fournir une telle description est

celle présentée dans [JM79], qui cherche à construire une grammaire régulière, à partir d'une grammaire d'entrée et du système de réécriture considéré.

Définition 2.2 (Grammaire régulière)**.** *Une grammaire régulière G est un quadruplet $\langle \Sigma, N, S, \Delta \rangle$ avec Σ un alphabet de symboles, appelés* terminaux*, N un alphabet de symboles (disjoints de Σ), appelés* non-terminaux*, $S \in N$ un non-terminal de départ avec $arity\,(S) = 0$, et Δ un ensemble de règles de production, présenté comme un TRS sur $\Sigma \cup N$.*

On dit qu'un terme t est produit par une grammaire G, s'il existe une dérivation de S à t : $S \Longrightarrow_{\Delta}^{*} t$. On note $\mathcal{L}(G)$ le langage ainsi engendré par G.

Étant donné un TRS $\mathcal{R}$ et une grammaire de départ G_0, l'approche proposée dans [JM79] consiste à construire une grammaire $G_{\mathcal{R}}$ sur-approximant l'ensemble des termes obtenus depuis G_0 par réécriture via $\mathcal{R}$:

$$\mathcal{R}^{*}(\mathcal{L}(G_0)) \subseteq \mathcal{L}(G_{\mathcal{R}})$$

Une version moderne de la méthode [JA07] présente un algorithme pour construire $G_{\mathcal{R}}$ en ajoutant de nouveaux non-terminaux et les règles de production associées dans G. Pour chaque règle de production générant un terme affecté par $\mathcal{R}$, on introduit un nouveau non-terminal représentant l'ensemble de ces termes, des nouveaux terminaux pour chaque sous-terme dans le membre droit affecté par $\mathcal{R}$, et pour chaque variable.

Exemple 2.2. *On considère la grammaire G_0, avec $\Sigma = \{z, s, even, odd, true, false\}$, $N = \{E, O, S\}$, et le système de règles de production Δ :*

$$\begin{cases} S & \rightarrow & even(E) \\ E & \rightarrow & z \\ E & \rightarrow & s(O) \\ O & \rightarrow & s(E) \end{cases}$$

Et on cherche à déterminer l'ensemble des termes obtenus suivant le TRS $\mathcal{R}$:

$$\begin{cases} even(z) & \rightarrow & true \\ even(s(n)) & \rightarrow & odd(n) \\ odd(z) & \rightarrow & false \\ odd(s(n)) & \rightarrow & even(n) \end{cases}$$

En d'autres termes, on cherche à démontrer que pour tout entier naturel n, défini dans l'algèbre de Peano, $even(n)$ n'est jamais réduit à $false$.

Ici, on a uniquement la règle de production $S \rightarrow even(E)$ à considérer : on introduit donc un non-terminal R_1 et les règles $S \rightarrow R_1$, $R_1 \rightarrow true$ pour la règle $even(z) \rightarrow true$ de $\mathcal{R}$, et un non-terminal R_2 et les règles $S \rightarrow R_2$, $R_2 \rightarrow odd(R_{2n})$ pour $even(s(n)) \rightarrow odd(n)$ et R_{2n} un non-terminal représentant la variable n dans cette règle, d'où $R_{2n} \rightarrow O$. On doit maintenant considérer la règle $R_2 \rightarrow odd(R_{2n})$: on introduit un non-terminal R_4 et les règles $R_2 \rightarrow R_4$, $R_4 \rightarrow even(R_{4n})$ pour la règle $odd(s(n)) \rightarrow odd(n)$ de $\mathcal{R}$, et R_{4n} un non-terminal représentant la variable n dans cette règle, d'où $R_{4n} \rightarrow E$. Enfin, à partir de la règle $R_4 \rightarrow even(R_{4n})$, on ajoute la règle $R_2 \rightarrow R_4$ d'après $even(s(n)) \rightarrow odd(n)$. On peut ainsi résumer le système de règles de

production obtenu :

$$\begin{cases} S & \twoheadrightarrow & even(E) \mid R_1 \mid R_2 \\ E & \twoheadrightarrow & z \mid s(O) \\ O & \twoheadrightarrow & s(E) \\ R_1 & \twoheadrightarrow & true \\ R_2 & \twoheadrightarrow & odd(R_{2n}) \mid R_4 \\ R_{2n} & \twoheadrightarrow & O \\ R_4 & \twoheadrightarrow & even(R_{4n}) \mid R_2 \\ R_{2n} & \twoheadrightarrow & E \end{cases}$$

On voit dans le système de règles de production de cette grammaire, qu'il n'y a bien aucune dérivation de S à false.

L'algorithme proposé itère sur les règles de production jusqu'à ce qu'aucun nouveau non-terminal et aucune nouvelle règle de production ne puisse être ajoutée à la grammaire. Cependant, comme il n'y a qu'un nombre fini de nouveau non-terminaux par règle du système $\mathcal{R}$, l'algorithme termine bien dans tous les cas. Cette approche est néanmoins uniquement dépendante de la grammaire de départ G_0 et du TRS $\mathcal{R}$, et ne permet donc pas d'affiner la sur-approximation quand celle obtenue est trop grossière.

Une méthode basée sur les automates d'arbres, présentée dans [GR10], propose un algorithme similaire par complétion d'automate, pour obtenir une sur-approximation plus fine du langage de sortie. Le formalisme d'automate d'arbres permet en effet de définir un filtrage sur les termes de l'algèbre pour décrire un sous-ensemble des termes comme le langage des termes filtrés par l'automate. La complétion d'automates d'arbres cherche à décrire la forme des termes d'un langage, initialement défini via un tel automate, transformés par réécriture selon un TRS donné.

Définition 2.3 (Automate d'arbres)**.** *Un automate d'arbres (ascendant) $\mathcal{A}$ est un quadruplet $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ avec $\mathcal{F}$ un alphabet de symboles, $\mathcal{Q}$ un ensemble d'états, $\mathcal{Q}_f \subseteq \mathcal{Q}$ un ensemble d'états finaux et Δ un ensemble de transitions de la forme $p \twoheadrightarrow q$ où $q \in \mathcal{Q}$ est un état et p un motif de $\mathcal{T}(\mathcal{C}, \mathcal{Q})$, appelé configuration.*

Un terme t est accepté par un automate d'arbres $\mathcal{A}$ si et seulement si $t \Longrightarrow_{\Delta}^{*} q$ avec $q \in \mathcal{Q}_f$, et le langage accepté par $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$, est l'ensemble des termes acceptés par $\mathcal{A}$.

Étant donné un automate d'arbres $\mathcal{A}_0$ et un TRS $\mathcal{R}$, le but de l'algorithme de complétion est donc de construire un automate $\mathcal{A}_{\mathcal{R}}$ sur-approximant l'ensemble des formes normales des termes de $\mathcal{L}(\mathcal{A}_0)$:

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_{\mathcal{R}})$$

Comme l'algorithme précédent, l'algorithme de complétion d'automate fonctionne en ajoutant des états et des transitions manquantes à l'automate. Mais il procède de manière beaucoup plus itérative, où chaque itération améliore l'approximation du langage, ce qui permet, en théorie, d'obtenir au final une exacte représentation de $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$.

Exemple 2.3. *On reprend le cas considéré dans l'Exemple 2.2. On peut représenter le même langage via l'automate $\mathcal{A}_0$ avec $\mathcal{F} = \{z, s, even, odd, true, false\}$, $\mathcal{Q} = \{q_{even}, q_{odd}, q_f\}$, $\mathcal{Q}_f = \{q_f\}$ et Δ représenté par le système :*

$$\begin{cases} z & \twoheadrightarrow & q_{even} \\ s(q_{even}) & \twoheadrightarrow & q_{odd} \\ s(q_{odd}) & \twoheadrightarrow & q_{even} \\ even(q_{even}) & \twoheadrightarrow & q_f \end{cases}$$

L'algorithme de complétion commence par détecter que, comme $even(z)$ *est accepté par l'automate initial* $\mathcal{A}_0$ *et que* $even(z) \Longrightarrow_{\mathcal{R}} true$*, on souhaiterait que true soit accepté par* $\mathcal{A}_{\mathcal{R}}$*. Pour se faire, on ajoute l'état* q_{true} *et les transitions* $true \to q_{true}$*,* $q_{true} \to q_f$ *à* $\mathcal{A}_{\mathcal{R}}$*. De façon similaire, on remarque que* $even(s(q_{odd})) \Longrightarrow_{\Delta}^{*} q_f$ *et* $even(s(q_{odd})) \Longrightarrow_{\mathcal{R}} odd(q_{odd})$*, donc on ajoute l'état* q_{co} *et les transitions* $odd(q_{odd}) \to q_{co}$*,* $q_{co} \to q_f$ *à* $\mathcal{A}_{\mathcal{R}}$*. On procède ainsi pour toutes les configurations acceptées par* $\mathcal{A}_0$ *et dérivés par* $\mathcal{R}$ *pour obtenir l'automate* $\mathcal{A}_1$*, et on recommence le processus en prenant en compte les états et transitions nouvellement ajoutés.*

Dans ce cas, on remarque qu'avec la nouvelle règle $odd(s(q_{even})) \Longrightarrow_{\Delta}^{*} q_f$*, il faut prendre en compte la dérivation* $odd(s(q_{even})) \Longrightarrow_{\mathcal{R}} even(q_{even})$*. Comme,* $even(q_{even}) \Longrightarrow_{\Delta} q_f$*, aucune nouvelle règle n'est nécessaire. L'algorithme s'arrête et on a* $\mathcal{A}_{\mathcal{R}} = \mathcal{A}_1$ *avec les transitions suivantes :*

$$\begin{cases} z & \to & q_{even} \\ s(q_{even}) & \to & q_{odd} \\ s(q_{odd}) & \to & q_{even} \\ even(q_{even}) & \to & q_f \\ true & \to & q_{true} \\ odd(q_{odd}) & \to & q_{co} \end{cases} \qquad \begin{array}{lcl} q_{true} & \to & q_f \\ q_{co} & \to & q_f \end{array}$$

Comme avec l'algorithme précédent, on voit ici que false n'est pas accepté par l'automate obtenu : pour tout entier n pair, $even(n)$ *ne peut pas être réduit à false.*

Comme l'algorithme de complétion d'automate procède par itérations améliorant la précision de l'automate au fur et à mesure, la terminaison n'est pas garantie : chaque pas d'itération peut introduire de nouveaux états et de nouvelles itérations, à partir des ajouts de l'itération précédente. Dans [Gen16], Genet identifie certaines classes de TRS pour lesquels la terminaison de l'algorithme est garantie.

Pour aider l'algorithme à construire un automate $\mathcal{A}_{\mathcal{R}}$ permettant de vérifier les propriétés recherchées, l'approche proposée dans [GR10] propose de construire une abstraction de la transformation. L'abstraction proposée est basée sur un système d'équations spécifiées par l'utilisateur, qui définissent des classes d'équivalence des configurations obtenues par réduction.

Exemple 2.4. *Dans les Exemples 2.2 et 2.3, les algorithmes proposés permettent de vérifier les propriétés recherchées parce que la distinction entre entier pairs et impairs est faite dans la grammaire/dans l'automate.*

Si maintenant on cherche à prouver que pour un entier n quelconque, $even(n + n)$ *ne peut pas être réduit à false, les deux approches échouent. Par exemple, on peut considérer l'automate* $\mathcal{A}_0$ *défini par le système de transition :*

$$\begin{cases} z & \to & q_n \\ s(q_n) & \to & q_n \\ double(q_n) & \to & q_f \end{cases}$$

avec le TRS $\mathcal{R}'$ *issu de* $\mathcal{R}$ *en ajoutant les règles :*

$$\begin{cases} plus(n, z) & \to & n \\ plus(n, s(m)) & \to & s(plus(n, m)) \\ double(n) & \to & even(plus(n, n)) \end{cases}$$

L'algorithme de complétion d'automates donne alors l'automate $\mathcal{A}_{\mathcal{R}}$ *défini par le système de*

transition :

$$\begin{cases} z & \to & q_n \\ s(q_n) & \to & q_n \\ double(q_n) & \to & q_f \\ plus(q_n, q_n) & \to & q_{sum} \\ s(q_{sum}) & \to & q_{sum} \\ even(q_{sum}) & \to & q_{even} \\ even(q_n) & \to & q_{even} \\ odd(q_{sum}) & \to & q_{odd} \\ odd(q_n) & \to & q_{odd} \\ true & \to & q_{true} \\ false & \to & q_{false} \end{cases} \qquad \begin{array}{lcl} q_{even} & \to & q_f \\ q_{odd} & \to & q_f \\ q_{true} & \to & q_f \\ q_{false} & \to & q_f \end{array}$$

On voit donc bien ici que l'automate obtenu accepte false. Néanmoins, en spécifiant l'équation $s(s(n)) = n$, l'algorithme peut reconnaitre deux classes d'équivalence regroupant respectivement les nombres paires et les nombres impaires. L'algorithme génère alors des états et des transitions correspondants à chacune de ces classes d'équivalence éliminant ainsi la dérivation permettant de reconnaître false.

Cette approche équationnelle peut également permettre de limiter la génération d'un nombre infini d'états en les fusionnant dans un nombre fini de classes d'équivalence. Genet montre ainsi dans [Gen16] que si le nombre de classes d'équivalences est fini alors l'algorithme de complétion proposé termine toujours.

Enfin, on peut observer que dans l'Exemple 2.4, on peut construire à partir de l'automate $\mathcal{A}_\mathcal{R}$ une abstraction du TRS $\mathcal{R}$ sur les configurations contenant la règle $even(q_n) \to q_{true}$. Or comme q_n reconnaît le terme $s(z)$, on peut ainsi construire un contre-exemple de l'abstraction obtenue (puisque $even(s(z))$ ne se réduit pas à *true*). L'approche proposée dans [HGJ20], propose alors, tant qu'on peut obtenir un tel contre-exemple, d'améliorer l'automate généré avec une procédure d'affinement de l'abstraction guidée par le contre-exemple (CEGAR [CGJ+00]). L'approche permet ainsi de limiter l'intervention de l'utilisateur en générant une abstraction plus fine sans, nécessairement, dépendre d'équations spécifiées par ce dernier.

2.3.2 Model-checking

Le *model-checking* est une approche consistant à construire une abstraction d'un programme ou d'un système (généralement appelé modèle) et de vérifier une propriété du programme en inspectant l'ensemble des états possibles donnés par le modèle. La méthode de vérification consistant donc à explorer l'ensemble des états du modèle, pour que cette dernière soit complète, il faut donc concevoir un modèle fini. Cela peut passer par une série de transformations qui doivent également être vérifié, pour garantir qu'elle préserve le comportement du système.

On présente ici quelques outils et méthodes de *model-checking*, mais on peut également noter que la problématique considérée peut donc également avoir des applications dans la conception de modèles.

Maude

On a déjà introduit Maude dans la Section 2.2.1 pour présenter son approche vis-à-vis de la réécriture stratégique. En pratique, Maude est un langage, basé sur la logique équationnelle et la logique de réécriture, destiné à la spécification et la vérification formelle de programme. Dans

ce but, Maude propose une approche de *model-checking* en vérifiant que l'ensemble des états atteignables définis par réécriture vérifient un invariant donné.

Si l'ensemble des états atteignables est infini, ou trop grand pour être explorés de façon complète, Maude propose également une approche de *model-checking* permettant de limiter le nombre de transitions pour atteindre les états considérés. En pratique, on préferera cependant construire un modèle fini par abstraction : Maude supportant la logique équationnelle, l'utilisateur peut définir un ensemble d'équations permettant de réduire l'ensemble infini d'états en une abstraction finie. Pour obtenir un modèle cohérent, il est également possible de vérifier que les équations préservent l'invariant recherché.

Higher order recursion scheme

Les *Higher Order Recursion Scheme* (HORS) sont une forme de grammaire d'arbres typés de l'ordre supérieur introduit dans [Ong06] :

Définition 2.4 (HORS)**.** *Un HORS $\mathcal{G}$ est un quadruplet $\langle \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ avec Σ un alphabet de symboles appelés* terminaux*, $\mathcal{N}$ un ensemble fini de symboles appelés* non-terminaux*, $S \in \mathcal{N}$ un non-terminal de départ typé o (pour* output*) et $\mathcal{R}$ un ensemble de règles de production de la forme $F\ x_1\ \cdots\ x_n \twoheadrightarrow t$ avec F un non-terminal de type $T_1 \mapsto \cdots \mapsto T_n \mapsto o$, $x_1, \ldots, x_n$ des variables de types respectifs $T_1, \ldots, T_n$, et t un motif.*

Cette famille de grammaires est particulièrement intéressante dans le contexte des méthodes de *model-checking*, puisque, comme montré dans [Ong06], la vérification de prédicats sur les termes générés par un HORS avec des types finis est décidable par *model-checking*. On peut cependant noter que la complexité théorique d'une telle vérification est k-EXPTIME avec k l'ordre du HORS (*i.e.* l'ordre maximal des non-terminaux).

L'approche par *model-checking* des HORSs n'étant cependant pas capable de gérer naturellement des types de données potentiellement infinis tels que des entiers ou des listes, des méthodes d'abstraction ont par la suite été introduites pour utiliser le formalisme d'HORS sur des problèmes plus généraux. L'approche présentée dans [KSU11] introduit ainsi un formalisme d'abstraction de prédicat : un programme en entrée est modélisé via une abstraction des variables numériques en booléens construits à partir de prédicats obtenus par une procédure CEGAR. L'approche a été implémentée dans le *model-checker* MoCHi [SKU$^+$20].

Une limitation majeure de l'utilisation d'HORSs classiques est que ces derniers ne supportent pas de primitive de filtrage par motif, ce qui ne permet donc pas d'implémenter d'opérations destructives. Ces approches ne sont donc naturellement pas adaptées pour modéliser des transformations de types algébriques. Une extension de la notion de HORS est proposée dans [KTU10], appelée *Higher-order Multi-parameter Tree Transducer* (HMTT) où les règles de production peuvent contenir des clauses de filtrage par motif de la forme :

$$match\ x \begin{cases} l_1 & \Longrightarrow & r_1 \\ \vdots & & \vdots \\ l_n & \Longrightarrow & r_n \end{cases}$$

où $l_1, \ldots, l_n$ sont des motifs de la forme $c\ y_1\ \cdots\ y_m$ avec c un symbole terminal d'arité m et $y_1, \ldots, y_m$ des variables. On peut, par exemple, définir le système $\mathcal{R}$ de l'Exemple 2.2 par le HMTT suivant :

$$\begin{array}{lcl} Even\ x & \twoheadrightarrow & match\ x \begin{cases} z & \Longrightarrow & true \\ s\ y & \Longrightarrow & Odd\ y \end{cases} \\ Odd\ x & \twoheadrightarrow & match\ x \begin{cases} z & \Longrightarrow & false \\ s\ y & \Longrightarrow & Even\ y \end{cases} \end{array}$$

A partir d'une abstraction des types d'entrées, exprimée par des automates d'arbres, il est alors possible de ramener la vérification d'invariants sur un tel HMTT à un problème sur un HORS fini.

Par la suite, [UTK10] propose une extension de la structure HMTT permettant d'introduire des annotations dans le programme modélisé. Chaque annotation est utilisée pour diviser le programme en HMTTs représentant les sous parties du programme avant et après l'annotation considérée. Cette approche permet également de contourner une limitation majeure du formalisme d'HMTT, qui est sa distinction forte des types d'arbres d'entrée et de sortie : seul un arbre du type d'entrée peut être "détruit" et seul un arbre du type de sortie peut être construit. En utilisant un formalisme d'automate pour exprimer ces annotations et les propriétés de vérification recherchées, ces prédicats servent d'abstraction des arbres d'entrée et de sortie de chaque HMTT ainsi généré. Il est alors possible de procéder à la vérification de la transformation modélisée par chaque HMTT, pour garantir la correction du programme d'origine.

Les approches basées sur le formalisme d'HMTT ne proposant cependant pas de procédure permettant de déduire automatiquement une abstraction adaptée pour leur vérification, on peut finalement mentionner l'approche introduite dans [OR11] qui présente une notion similaire d'HORSs avec des primitives de filtrage par motif, appelés *Pattern Matching Recursion Scheme* (PMRS). L'intérêt de cette approche, comparé aux HMTTs, est quelle introduit un mécanisme d'abstraction automatique basée sur une procédure de type CEGAR. Ce mécanisme n'est cependant pas complet : pour un PMRS représentant le problème traité dans l'Exemple 2.4, la procédure échoue à construire une abstraction distinguant entiers paires et impaires.

2.3.3 Typage et annotations

De façon similaire à l'approche de Nanopass qui permet de garantir une passe comme une transformation entre un langage de départ et un langage d'arrivée, le typage statique d'un programme est une méthode courante d'analyse et de vérification. L'utilisation de système de types complexes permet ainsi de vérifier statiquement, qu'à chaque pas d'exécution, l'état courant du programme est en accord avec les spécifications décrites par le système de type donné. De nombreuses approches ont ainsi proposé d'exprimer des systèmes de types de plus en plus complexes, permettant de décrire de façon aussi précise que possible des propriétés de corrections.

Pour revenir à l'approche de Nanopass, la présence ou l'absence de certains symboles peut être décrite par l'utilisation de *sous-types constructeurs* [BF99]. Cette approche propose un formalisme introduisant une relation de sous-typage entre deux types dont les ensembles des symboles constructeurs sont reliés par une relation d'inclusion. En pratique, une telle approche permet ainsi de construire des systèmes de types complexes décrivant des garanties syntaxiques similaires à celles proposées par Nanopass.

Pour faciliter l'utilisation de ce genre de formalisme, une approche classique consiste à décorer les types de bases avec des annotations permettant de décrire des propriétés et des prédicats du programme considéré. C'est l'approche considéré par la notion de *Refinement Types* [FP91] qui propose un formalisme d'annotations de types permettant d'introduire des prédicats dans les signatures de type. L'approche a ainsi été implémentée dans de nombreux langages, tels qu'Agda et Coq. On peut, par exemple, annoter la signature de la fonction `head` (qui renvoie le premier élément d'une liste) pour spécifier qu'elle ne s'applique que sur les listes non-vides, plutôt que de dépendre d'une vérification à l'exécution :

```
{-@ head :: {l:[a] | length l > 0} -> a @-}
head (x :: _) = x
```

Par la suite, le développement de *Liquid Types* [RKJ08] propose d'introduire des mécanismes d'inférence sur un système d'annotations de type basé sur les *Refinement types*. Cette inférence permet de déduire certaines annotations à partir de celles données par l'utilisateur, permettant ainsi de limiter la dépendance aux contributions manuelles de l'utilisateur.

Même si la notion de *Recursive Refinements* [KRJ09] a permis, par la suite, l'écriture de prédicats prenant en compte la forme et les sous-termes de types inductifs, cette approche est principalement destiné à l'expression et la vérification de propriétés sur des entiers ou des types intégrés à la sémantique d'annotation (potentiellement indépendante de celle du langage lui-même). En effet, l'utilisation de *Refinement Types* permet principalement la description de propriétés locales des termes, et n'est donc pas adaptée à l'écriture et la vérification de propriétés structurelles syntaxiques des types algébriques.

2.3.4 Vérification de transformations XML

La vérification de programmes de traitement de documents XML est un également un domaine ayant bénéficier du développement de nombreuses méthodes permettant la vérification de propriétés syntaxiques sur des transformations d'arbres. En effet, c'est un domaine qui a de très nombreuses applications dans l'industrie avec des outils bien établis tels que les langages XSLT ou XQuery. L'idée consiste à traiter un document XML pour en extraire de l'information et/ou générer un document avec un schéma XML différent, ou un autre format (*e.g.* HTML).

Un document XML pouvant être représenté comme un arbre typé par le schéma XML spécifié, ce type de transformation XML est en général vérifié par des approches de *type-checking*. Des approches telles que celles présentées dans [Toz06] et [MSV03] proposent d'utiliser un formalisme de transducteurs d'arbres pour modéliser les programmes de transformation considérés. Il est alors possible de typer la transformation pour vérifier les structures d'entrée et de sortie du transducteur. L'utilisation de transducteurs rend aussi la vérification via *model-checking* possible, comme dans [KTU10] présenté précédemment.

D'autres approches, telles que XDuce [HP03] ou CDuce [BCF03], ont introduit des formalismes d'annotation de type permettant d'exprimer des spécifications supplémentaires à garantir sur le programme de transformation. Le type annoté est ensuite vérifié avec une méthode de sous-typage par sémantique [FCB02] : l'idée consiste à construire une notion de sémantique sur les termes et les types annotés et d'exprimer la relation de sous-typage comme une inclusion de sémantique.

2.4 Synthèse

On propose donc, dans ce mémoire, de définir un formalisme permettant la spécification et la vérification de transformations d'arbres syntaxiques. On s'intéresse, en particulier, à garantir des propriétés syntaxiques sur le résultat de la transformation considérée, pour montrer qu'un motif donné est absent de ce résultat. Pour cela, on propose de s'appuyer sur une approche algébrique utilisant le formalisme de réécriture.

La notion de réécriture est en effet un formalisme très répandu dans le domaine des méthodes formelles, notamment pour son expressivité puisqu'il permet de définir de façon très naturelle des transformations sur des types de données algébriques. Il est tout particulièrement adapté pour encoder des programmes fonctionnels (y compris dans le cas de programmes à l'ordre supérieur [Joh85]). De plus, des approches basées sur la réécriture stratégique ou la génération de règles de réécriture [CLM15] facilitent grandement l'écriture d'un système de réécriture décrivant le comportement de la transformation étudiée. De nombreux langages, tels que Tom, Stratego,

Rascal ou Maude, se sont ainsi basés sur la réécriture pour proposer des outils de spécification et/ou d'analyse de programmes.

Du point de vue de la vérification formelle, la réécriture est utilisée par certaines approches telle que la complétion d'automates [Gen14], qui propose de construire un automate d'arbres pour décrire le langage des termes obtenus par réécriture suivant un système donné. Des approches alternatives, telles que le *model-checking* d'HORS [Ong06] ou l'utilisation de systèmes de types annotés [FP91, RKJ08], proposent de garantir certaines propriétés sur les résultats d'un programme en utilisant des formalismes plus compliqués à prendre en main et/ou moins adaptés à des types de données algébriques.

3

Exemption de Motif et Sémantique

La problématique considérée étant de vérifier que des programmes de transformation d'arbres produisent des résultats ne contenant pas certains motifs algébriques, on cherche à proposer un formalisme, basé sur l'algèbre de termes et la réécriture, permettant de vérifier que les formes normales potentiellement issues d'une relation de réécriture ne contiennent pas les motifs exclus, *i.e.* qu'aucun sous-terme de ces formes normales ne sont filtrés par ces motifs.

Dans le contexte des langages fonctionnels, les transformations sont réalisées par le biais de fonctions. Le formalisme proposé doit donc permettre de vérifier des garanties quant à la forme des résultats de ces fonctions. En pratique, le système de type considéré et la signature de type des fonctions fournissent déjà de l'information sur les symboles constituant le terme obtenu. Cependant, quand il s'agit de prouver qu'un symbole ou un motif sera absent du résultat, ces informations ne sont généralement pas suffisantes.

Le formalisme considéré propose donc d'annoter les symboles de fonction (représentés par les symboles définis $\mathcal{D}$ en algèbre de termes) par des profils détaillant le comportement attendu de la fonction associée vis-à-vis des propriétés syntaxiques recherchées. A partir de ces annotations, on étend la notion de sémantique close introduite dans [CM19], aux termes contenants des symboles définis, pour représenter une sur-approximation de l'ensemble des formes normales attendues. Étant donné un CBTRS encodant ces fonctions, on pourra vérifier que cette sémantique est en accord avec la relation de réécriture.

Dans un premier temps, on introduit la notion d'Exemption de Motif qui définit formellement la propriété syntaxique recherchée et le système d'annotation utilisé. Dans la Section 3.2, on étend la notion de sémantique close aux termes contenant des symboles définis en sur-approximant l'ensemble des formes normales attendues en se basant sur le comportement spécifié par les annotations des symboles définis. Enfin, on étudiera les propriétés de préservation de cette sémantique via une relation de réécriture, afin de vérifier que les annotations données sont cohérentes avec le CBTRS encodant le comportement de la transformation étudiée.

3.1 Exemption de Motif

On cherche donc à proposer un formalisme basé sur une notion d'annotation spécifiant le comportement attendu de la réduction associé à un symbole défini représentant l'application d'une fonction de transformation. Le but final étant de pouvoir prouver l'absence de sous-terme filtré par un motif donné, on introduit dans un premier une propriété définissant formellement cette absence de motif, puis on utilisera cette notion pour définir le système d'annotations considéré.

3.1.1 Termes constructeurs exempts

On définit donc, pour les valeurs, la propriété d'Exemption de Motifs pour donner une description syntaxique de la forme des termes obtenus comme résultat de la transformation considérée :

Définition 3.1 (Valeur exempte)**.** *Étant donné un motif étendu p, on dit qu'une valeur $v \in \mathcal{T}(\mathcal{C})$ est exempte de p si et seulement si $\forall \omega \in \mathcal{P}os(v), p \not\lessdot v_{|\omega}$.*

Une valeur est exempte d'un motif p si et seulement p ne filtre aucun sous-terme de la valeur. On pourra noter que, comme $\perp$ ne filtre rien, toute valeur est exempte de $\perp$.

Cette propriété d'exemption peut bien sûr être utilisée dans le contexte de la transformation de programmes, où elle est particulièrement bien adaptée pour décrire le résultat d'une passe de compilation qui aura un effet limité sur les constructions du langage. On retrouve donc comme exemple naturel le cas de l'élimination de sucre syntaxique.

Exemple 3.1. *On considère le langage de λ-expression, introduit dans la Section 2.1, composé de constructions λ, application, let et de variables. On a vu dans la Section 2.1, la construction let peut être vue comme un sucre syntaxique.*

Une expression de ce langage peut être représentée par un terme de l'algèbre définie par la signature $\Sigma_{let} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$
\begin{array}{rcl}
\mathsf{Expr} & := & \mathit{lambda}(\mathsf{Int}, \mathsf{Expr}) \\
& | & \mathit{apply}(\mathsf{Expr}, \mathsf{Expr}) \\
& | & \mathit{let}(\mathsf{Int}, \mathsf{Expr}, \mathsf{Expr}) \\
& | & \mathit{var}(\mathsf{Int})
\end{array}
\qquad\qquad
\begin{array}{rcl}
\mathsf{Int} & := & z \\
& | & s(\mathsf{Int})
\end{array}
$$

Les variables sont ici identifiées par un entier de Peano.

Dans ce langage, plutôt que de représenter les termes obtenus par élimination du sucre syntaxique let comme des termes d'une sous-sorte comme proposée dans la Section 2.1, on peut les représenter comme les valeurs de sorte Expr *exemptes du motif $let(i, e_1, e_2)$.*

Mais cette propriété peut facilement s'adapter pour décrire la forme souhaitée de formules logiques (comme pour la forme normale négative dans l'Exemple 3.2), des propriétés de sécurité, ou encore, en utilisant des notions d'interprétation abstraite, des propriétés plus compliquées (comme pour les listes triées dans l'Exemple 3.3).

Exemple 3.2. *On considère des formules logiques composées de prédicats, pouvant dépendre de variables quantifiées par des quantificateurs universels ou existentiels, et des opérateurs de conjonction, disjonction, implication et négation. On dit qu'une telle formule est en forme normale négative si l'opérateur de négation n'est appliqué que sur des prédicats et que la formule ne contient pas d'implication.*

Les formules logiques considérées peuvent être représentées par les termes de l'algèbre définie par la signature $\Sigma_{formula} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$
\begin{array}{rcl}
\mathsf{Int} & := & z \\
& | & s(\mathsf{Int})
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{Var} & := & \mathit{var}(\mathsf{Int})
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{VarList} & := & \mathit{nil} \\
& | & \mathit{cons}(\mathsf{Var}, \mathsf{VarList})
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{Formula} & := & \mathit{predicate}(\mathsf{Int}, \mathsf{VarList}) \\
& | & \mathit{not}(\mathsf{Formula}) \\
& | & \mathit{and}(\mathsf{Formula}, \mathsf{Formula}) \\
& | & \mathit{or}(\mathsf{Formula}, \mathsf{Formula}) \\
& | & \mathit{impl}(\mathsf{Formula}, \mathsf{Formula}) \\
& | & \mathit{exists}(\mathsf{Var}, \mathsf{Formula}) \\
& | & \mathit{forall}(\mathsf{Var}, \mathsf{Formula})
\end{array}
$$

Les variables propositionnelles et les prédicats sont ici identifiés par un entier de Peano.

Dans ce contexte, les formules en forme normale négative sont donc les valeurs de sorte Formula *exemptes du motif* $impl(f_1, f_2) + not(and(f_1, f_2)) + not(or(f_1, f_2)) + not(forall(v, f)) + not(exists(v, f)) + not(not(f))$*, qu'on pourra noter de façon plus concise par* $p_{nnf} = impl(f_1, f_2) + not(!predicate(i, l))$.

Exemple 3.3. *On considère des listes d'expressions constituées des constantes* 0 *et* 1*, ordonnées avec* $0 < 1$*. Dans ce contexte, une liste triée est donc une liste qui ne contient aucune constante* 0 *après une constante* 1*.*

De plus, on peut représenter ces listes comme les termes de sorte List *de l'algèbre définie par la signature* $\Sigma_{sorted} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ *décrite par les types algébriques :*

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{Expr} & := & A & \mathsf{List} & := & nil \\ & | & B & & | & cons(\mathsf{Expr}, \mathsf{List}) \end{array}$$

où le constructeur A*, respectivement* B*, représente la constante* 0 *respectivement* 1 *(on a donc* $A < B$*).*

On peut ainsi représenter les listes triées par les valeurs de sorte List *exemptes du motif* $p_{sorted} = cons(B, cons(A, l))$.

La notion d'Exemption de Motif s'étend également naturellement aux termes constructeurs de l'algèbre en considérant toutes les instances du terme, *i.e.* en instanciant ses variables avec des valeurs :

Définition 3.2 (Terme constructeur exempt)**.** *Étant donné un motif étendu* p*, on dit qu'un terme constructeur* $u \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ *est exempt de* p *si et seulement, pour toute substitution* σ *telle que* $\sigma(u) \in \mathcal{T}(\mathcal{C})$*,* $\sigma(u)$ *est exempt de* p*.*

Un terme constructeur est donc exempt d'un motif si et seulement si toutes ses instances valeurs le sont. Comme pour les valeurs exemptes, étant donné que le motif $\bot$ ne filtre rien, tout terme constructeur est exempt de $\bot$.

Cette généralisation garantit la stabilité de l'Exemption de Motif par substitution, mais on peut naturellement observer qu'elle est également préservée pour les sous-termes :

Proposition 3.1. *Étant donné un motif étendu* p *et un terme* $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$*,* t *est exempt de* p *si et seulement si* $t_{|\omega}$ *est exempt de* p*, pour toute position* $\omega \in \mathcal{P}os(t)$*.*

Démonstration. Soit p un motif étendu et t un terme de $\mathcal{T}(\mathcal{C}, \mathcal{X})$.

Si $t_{|\omega}$ est exempt de p pour toute position $\omega \in \mathcal{P}os(t)$, alors, en particulier, $t_{|\epsilon} = t$ est exempt de p.

On suppose maintenant que t est exempt de p et on montre l'implication directe par induction sur la forme t :

- Si $t = x \in \mathcal{X}$ ou $t = c \in \mathcal{C}^0$, alors $\mathcal{P}os(t) = \{\epsilon\}$ et la preuve est immédiate.
- Si $t = c(t_1, \ldots, t_n)$ avec $n > 0$, comme t est exempt de p, pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{C})$, on a alors $\sigma(t)$ exempt de p. Donc, pour tout $i \in [1, n]$, t_i est exempt de p, car sinon il existerait une substitution σ avec $\sigma(t_i) \in \mathcal{T}(\mathcal{C})$, et une position $\omega \in \mathcal{P}os(\sigma(t_i))$ telles que $p \not\ll \sigma t_{i|\omega}$, et donc il existerait une substitution σ', généralisant σ sur t, tel que $\sigma'(t) \in \mathcal{T}(\mathcal{C})$ et $p \not\ll \sigma'(t)_{|i.\omega}$, *i.e.* $\sigma'(t)$ non-exempt de p. Par induction, on a donc $t_{|\omega}$ exempt de p pour toute position $\omega \in \mathcal{P}os(t)$ ($\omega \neq \epsilon$ par induction, et $\omega = \epsilon$ puisque $t_{|\epsilon} = t$).

□

Enfin, on observe que la propriété d'Exemption de Motif peut se composer via l'opérateur de disjonction + des motifs étendus :

Proposition 3.2. *Étant donné des motifs étendus p_1, p_2, et un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, t est exempt de $p_1 + p_2$ si et seulement si t est exempt de p_1 et de p_2*

Démonstration. La preuve est immédiate en observant que pour toute valeur $v \in \mathcal{T}(\mathcal{C})$ et pour tous motifs étendus p_1, p_2, on a $p_1 + p_2 \not\lhd v$ si et seulement $p_1 \not\lhd v$ et $p_2 \not\lhd v$. □

L'Exemption de Motif étant ainsi une propriété syntaxique à la fois préservée par la relation de sous-terme et par substitution, elle est particulièrement bien adaptée à un formalisme basé sur la réécriture. En effet, une relation de réécriture opère en instanciant certains sous-termes de départ (cf. Définition 1.20). On proposera par la suite une généralisation de cette notion à l'ensemble des termes de l'algèbre, mais pour cela il est nécessaire de pouvoir caractériser les symboles définis (qui sont les seuls à ne pas être déjà pris en compte dans les définitions précédentes) en termes d'Exemption de Motif.

3.1.2 Profils et annotations

En algèbre de termes, une fonction de transformation est représentée par un symbole défini de $\mathcal{D}$, ce qui permet de décrire le comportement de la transformation associée par la relation de réécriture induite d'un CBTRS. Le formalisme de réécriture permet ainsi de considérer la potentielle forme normale des termes obtenue par réduction, pour lesquels on cherche donc à garantir des propriétés d'Exemption de Motif (Définition 3.1). Afin de pouvoir obtenir de telles garanties, les propriétés d'Exemption de Motif peuvent être utilisées comme un moyen d'enrichir la signature de sorte des symboles définis.

En effet, de façon similaire aux approches de raffinement de type, on propose d'annoter les symboles définis de la signature considérée avec des profils indiquant les propriétés d'Exemption de Motif attendues pour la forme normale potentielle d'un terme ayant le symbole défini considéré comme symbole de tête. Ainsi un symbole $\varphi : s_1 * \cdots * s_n \mapsto s$ pourra être annoté par un ou plusieurs profil(s) de la forme $p_1 * \cdots * p_n \mapsto p$ avec $p_1, \ldots, p_n, p$ des motifs linéaires. Un tel profil indiquera qu'un terme ayant φ comme symbole de tête, avec comme argument des termes respectivement exempts de $p_1, \ldots, p_n$ devrait, au final, être réduit en une valeur exempte de p ; ou plus précisément, que pour un terme clos de la forme $t = \varphi(t_1, \ldots, t_n)$, si $t_1, \ldots, t_n$ ne peuvent être réduits qu'en des termes respectivement exempts de $p_1, \ldots, p_n$, alors la potentielle forme normale de t devra être une valeur exempte de p.

On considère donc que tous les symboles définis $\varphi \in \mathcal{D}$ sont annotés par un ensemble de profils $\mathcal{P}$, et on notera le symbole $\varphi^{!\mathcal{P}}$ pour expliciter cette annotation. On pourra également omettre l'annotation pour alléger les notations. On peut remarquer que même dans le cas où $\mathcal{P} = \emptyset$, on aura toujours $t = \varphi(t_1, \ldots, t_n)$ réduit en un terme exempt de $\bot$ si et seulement si $t_1, \ldots, t_n$ sont exempts de $\bot$, puisque tout terme est exempt de $\bot$. Cela revient à considérer un profil $\bot * \cdots * \bot \mapsto \bot$ par défaut, que l'on peut donc omettre de l'annotation $\mathcal{P}$.

Exemple 3.4. *L'exemple 3.2 a introduit une algèbre, définie à partir de la signature $\Sigma_{formula}$, permettant de représenter des formules logiques et explique comment on peut décrire la forme normale négative de ces formules comme une propriété d'Exemption du Motif $p_{nnf} = impl(f_1, f_2) + not(!predicate(i, l))$.*

A partir de cette observation, on peut donc ajouter le symbole défini $nnf^{!\mathcal{P}}$: $\mathsf{Formula} \mapsto \mathsf{Formula} \in \mathcal{D}$ à la signature $\Sigma_{formula}$ pour représenter le symbole de fonction transformant une

formule quelconque en une formule équivalente en forme normale négative. Le système de réécriture décrivant le comportement de la fonction associée au symbole est donnée en Annexe C, et on peut annoter le symbole avec $\mathcal{P} = \{\bot \mapsto p_{nnf}\}$.

Le profil $\bot \mapsto p_{nnf}$ *indique, en effet, que tout terme dont le symbole de tête est nnf doit être réduit en un terme représentant une formule en forme normale négative, i.e. pour toute valeur* v : Formula*, on voudra que la forme normale de* $nnf(v)$ *soit une valeur exempte de* p_{nnf}.

Ces profils peuvent ainsi être découpés en deux : le membre gauche fournit des pré-conditions s'appliquant sur les arguments de la fonction, *i.e.* sur les termes en argument du symbole défini, tandis que le membre droit donne une post-condition à satisfaire pour le résultat de la fonction, *i.e.* la potentielle forme normale obtenue par réduction suivant un CBTRS. Dans l'exemple précédent, la pré-condition est donc vide, ce qui est symbolisé par une Exemption du motif $\bot$ (qui est toujours vérifiée), mais ce n'est pas toujours le cas. En effet, une approche initiale, présentée dans [CLM20a], proposait de simplement annoter les symboles définis avec un unique motif indiquant la propriété d'Exemption de Motif à vérifier par la potentielle forme normale du terme considérée, *i.e.* la post-condition (sans pré-condition). Cependant, cette approche est naturellement limitée par une vérification uniquement basée sur le résultat de la réduction sans prendre en compte les informations disponibles sur les arguments fournis. De plus, de tels profils peuvent toujours être représentés avec l'approche présentée ici en considérant des profils avec des pré-condition vide, via l'Exemption du motif $\bot$, comme dans l'exemple précédent.

Exemple 3.5. *Considérons des expressions pouvant être un entier ou une liste de telles expressions. Ce langage d'expression peut être représenté par l'algèbre de termes définie par la signature* $\Sigma_{list} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ *décrite par les types algébriques :*

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} \mathsf{Expr} & := & int(\mathsf{Int}) & \mathsf{Int} & := & z & \mathsf{List} & := & nil \\ & | & lst(\mathsf{List}) & & | & s(\mathsf{Int}) & & | & cons(\mathsf{Expr}, \mathsf{List}) \end{array}$$

Dans cette algèbre, on peut définir les listes plates, i.e. les listes ne contenant pas de liste imbriquée, avec les valeurs de sorte List *exempte du motif* $p_{flat} = cons(lst(l_1), l_2)$.

On introduit donc le symbole défini $flatten^{!\mathcal{P}_1} : \mathsf{List} \mapsto \mathsf{List} \in \mathcal{D}$ *représentant la fonction d'aplatissement d'une Liste, i.e. la fonction prenant une liste en argument et renvoyant une liste plate contenant tous les entiers contenus dans la liste d'entrée et ses éventuelles listes imbriquées, et un symbole défini* $concat^{!\mathcal{P}_2} : \mathsf{List} * \mathsf{List} \mapsto \mathsf{List}$ *représentant la fonction de concaténation de deux listes. Naturellement on peut donc annoter le symbole flatten avec* $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ *pour indiquer la garantie souhaitée que la forme normale d'un terme ayant pour symbole de tête flatten doit être une liste plate (sans pré-condition sur l'argument). Mais, on peut également observer que la concaténation de deux listes plates est censée être une liste plate : dans ce cas, le membre gauche du profil permet bien de décrire la pré-condition exigeant aux deux listes d'être plates, i.e. exemptes de* p_{flat}*. On peut donc annoter concat avec* $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$*, pour expliciter que pour toutes listes* $l_1, l_2 \in \mathcal{T}_{\mathsf{List}}(\mathcal{C})$*, si* l_1 *et* l_2 *sont exemptes de* p_{flat}*, la forme normale de* $concat(l_1, l_2)$ *devra également être exempte de* p_{flat}.

Bien que cette dernière annotation ne soit pas forcément utile pour expliciter la garantie d'aplatissement via la fonction flatten, suivant le système de réécriture décrivant sa réduction, elle peut être nécessaire à sa vérification.

En effet, les profils fournissent donc une indication des propriétés d'Exemption de Motif attendues sur les formes normales potentiellement obtenues par réduction suivant une relation de réécriture définie par un CBTRS. On va donc chercher à prouver que le système considéré est bien cohérent avec ces annotations. Pour cela, on va, dans un premier temps, donner une

généralisation de la propriété d'Exemption de Motif à l'ensemble des termes de l'algèbre, donc aux termes contenant également des symboles définis.

3.1.3 Généralisation de l'Exemption de Motif

Les profils introduits précédemment viennent donc enrichir le système de type pour donner de l'information sur la forme attendue d'un terme après sa réduction. Ces informations supplémentaires peuvent être utilisées pour définir une sur-approximation de la potentielle forme normale du terme considéré : pour chaque sous-terme ayant un symbole défini comme symbole de tête, on peut sur-approximer la forme normale attendue de ce sous-terme par n'importe quelle valeur respectant les propriétés d'Exemption de Motif imposées par le profil.

On peut ainsi se servir de cette sur-approximation pour donner une généralisation de la propriété d'Exemption de Motif, telle qu'un terme contenant des symboles définis vérifie une propriété d'Exemption de motif si et seulement tous les termes de la sur-approximation la vérifie également :

Définition 3.3 (Exemption de Motif)**.** *Étant donné un motif étendu p, on dit que :*

- *une valeur $v \in \mathcal{T}(\mathcal{C})$ est exempte de p si et seulement $\forall \omega \in \mathcal{P}os(v), p \not\lhd v_{|\omega}$;*
- *un terme constructeur $u \in \mathcal{T}(\mathcal{C}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{C})$ est exempt de p si et seulement, pour toute substitution σ telle que $\sigma(u) \in \mathcal{T}(\mathcal{C})$, $\sigma(u)$ est exempt de p ;*
- *un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{C}, \mathcal{X})$ est exempt de p si et seulement si, pour toute position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et pour toute valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$, le terme $t\,[v]_\omega$ est exempt de p.*

où $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$ est le motif annotant *de $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$, défini par $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n) = \sum_{q \in \mathcal{Q}} q$ avec $\mathcal{Q} = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ t.q. $\forall i \in [1, n], t_i$ est exempt de $l_i\}$*

Pour les termes constructeurs, la définition est identique aux définitions originellement données dans la Section 3.1.1 : une valeur est exempte d'un motif p si et seulement si p ne filtre aucun de ses sous-termes, et un terme constructeur est exempt de p si et seulement si toutes ses instances valeurs le sont. Enfin, on a donc qu'un terme contenant des symboles définis est exempt de p si et seulement si remplacer les sous-termes ayant un symbole définie $\varphi_s^{!\mathcal{P}} \in \mathcal{D}$ comme symbole de tête par une valeur de sorte s exempt du *motif annotant*, résultant de l'annotation $\mathcal{P}$ et de ses sous-termes respectifs, donne également un terme exempt de p. Pour chaque sous-terme de la forme $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$, et chaque profil $l_1 * \cdots * l_n \mapsto r$ tel que $t_1, \ldots, t_n$ sont respectivement exempts de $l_1, \ldots, l_n$, $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ est donc exempt de r.

Exemple 3.6. *On considère l'algèbre de termes introduite dans l'Exemple 3.5. On note $p_{flat} = cons(lst(l_1), l_2)$, et on rappelle que les symboles définis sont annotés $flatten^{!\mathcal{P}_1}$ et $concat^{!\mathcal{P}_2}$ avec $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ et $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$.*

- *On remarque que $cons(int(z), nil)$ est exempt de p_{flat}, puisque c'est une valeur et que p_{flat} ne filtre aucun de ses sous-termes ;*
- *On peut également remarquer $cons(int(i), nil)$ est exempt de p_{flat}, puisque quelque soit l'instanciation donnée à la variable i, la valeur obtenue est exempte de p_{flat} ;*
- *Cependant, on peut observer que $cons(int(i), cons(lst(nil), nil))$ n'est pas exempt de p_{flat} puisque le sous terme $cons(lst(nil), nil)$ est filtré par p_{flat} ;*
- *On peut donc en déduire que le terme $cons(int(i), l)$ n'est pas exempt de p_{flat}, puisque la variable l peut être instanciée par $cons(lst(nil), nil)$;*

- *En revanche, $flatten^{!\mathcal{P}_1}(l)$ est exempt de p_{flat}, puisque comme l est exempt de $\bot$, on a le motif annotant $\triangleright^{\mathcal{P}_1}(l) = p_{flat}$;*
- *Ainsi, on a également $cons(int(i), flatten^{!\mathcal{P}_1}(l))$ exempt de p_{flat}, puisque pour toute valeur v de sorte* List *exempte p_{flat}, $cons(int(i), v)$ est exempt de p_{flat} ;*
- *Mais, $cons(e, flatten^{!\mathcal{P}_1}(l))$ n'est pas exempt de p_{flat}, puisqu'étant donnée une valeur v de sorte* List *exempt de p_{flat}, on peut instancier la variable e par $lst(nil)$ et $cons(lst(nil), v)$ n'est pas exempt de p_{flat} ;*
- *Enfin, on remarque que, comme $\triangleright^{\mathcal{P}_2}(l_1, l_2) = \bot$, le terme $concat^{!\mathcal{P}_2}(l_1, l_2)$ n'est pas exempt de p_{flat}, mais en revanche, comme $\triangleright^{\mathcal{P}_2}\left(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2)\right) = p_{flat}$, le terme $concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))$ est exempt de p_{flat}.*

En prenant en compte tous les profils de l'annotation $\mathcal{P}$, on sur-approxime ainsi la forme normale du sous terme $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ par l'ensemble des valeurs vérifiant toutes les propriétés d'Exemption de Motif attendues. En répétant l'opération pour tous les sous-termes ayant un symbole défini comme symbole de tête, on obtient donc une sur-approximation de la forme normale du terme considéré.

Comme on l'a vu dans la Section 2.2.3, étant donné un motif étendu linéaire p, on peut construire un motif additif r tel que toute valeur est filtrée par p si et seulement si elle est filtrée par r. L'exemption de p est alors équivalente à l'exemption de r. Par la suite on ne considèrera donc des Propriétés d'Exemption de Motif que sur des motifs additifs linéaires.

Les propriétés prouvées précédemment, comme la composition d'Exemption de Motif via l'opérateur de disjonction + des motifs étendus (Proposition 3.2) s'appliquent également à cette généralisation. On remarque que la préservation de l'Exemption de Motif est aussi vérifiée pour les termes linéaires :

Proposition 3.3. *Étant donnés un motif étendu p et un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ linéaire, t est exempt de p si et seulement si $\sigma(t)$ est exempt de p, pour toute substitution σ.*

Démonstration. Soient $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ un terme linéaire, on prouve par induction sur k le nombre de symboles définis dans t que pour tout motif étendu p, t est exempt de p si et seulement si $\sigma(t)$ est exempt de p, pour toute substitution σ.

On commence par montrer l'implication indirecte : on suppose que $\sigma(t)$ est exempt de p pour toute substitution σ, et on montre que t est exempt de p.

- Si $k = 0$, alors $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ et, comme $\sigma(t)$ est exempt de p pour toute substitution σ, on a, notamment, $\sigma(t)$ est exempt de p pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{C})$. Par définition de la propriété d'Exemption de Motif, t est donc exempt de p.
- On considère maintenant $k > 0$ tel que pour tout terme $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ linéaire ayant strictement de k symbole défini, pour tout motif étendu q, si $\sigma(u)$ est exempt de q pour toute substitution σ, alors u est exempt de q. Comme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on cherche à montrer que pour toute position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ où $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et pour toute valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$, $t\,[v]_\omega$ est exempt p. Soient ω une telle position et v une telle valeur, on note $u := t\,[v]_\omega$. Pour toute substitution σ, en notant $\sigma' := \{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma) \setminus \mathcal{V}ar(t_{|\omega})\}$, on observe que $\sigma(u) = \sigma'(t)\,[v]_\omega$. Or, par hypothèse, $\sigma'(t)$ est exempt de p, donc, par définition de l'Exemption de Motif, $\sigma(u)$ est exempt de p, pour toute substitution σ. D'après l'hypothèse d'induction, on a alors u exempt de p, et on peut donc conclure que t exempt de p.

Par induction sur k, on a donc bien démontré l'implication indirecte.

On prouve maintenant l'implication directe par contraposée [1] : on considère un motif étendu p et une substitution σ tels que $u := \sigma(t)$ n'est pas exempt de p, et on montre que t n'est pas exempt de p. On suppose, dans un premier temps, que $k = 0$, *i.e.* $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, et on prouve par induction sur m le nombre de symboles définis dans u qu'il existe une substitution σ' telle que $\sigma'(t)$ est une valeur non-exempte de p.

- Si $m = 0$, alors $u \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. Si $u \in \mathcal{T}(\mathcal{C})$, alors σ est une substitution telle que $\sigma(t)$ est une valeur non-exempte de p. Sinon, par définition de l'Exemption de Motif, il existe une substitution σ' telle que $\sigma'(u)$ est une valeur non-exempt de p. On a donc $\sigma' \circ \sigma$ une substitution telle que $\sigma' \circ \sigma(t)$ est une valeur non-exempte de p.
- On considère maintenant $m > 0$ tel que pour toute substitution σ' avec $\sigma'(t)$ ayant strictement moins de m symboles définis, il existe une substitution σ'' telle que $\sigma''(t)$ est une valeur non-exempte de p.
  Comme u possède au moins un symbole défini et n'est pas exempt de p, il existe une position $\omega \in \mathcal{P}os(u)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$, le terme $u[v]_\omega$ n'est pas exempt de p. Comme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, il existe une position $\omega' < \omega$ telle que $t_{|\omega'} = x \in \mathcal{X}$. On note $\omega = \omega'.\omega''$, et on définit σ' la substitution telle que $\mathcal{D}om(\sigma') = \mathcal{D}om(\sigma)$ avec $\sigma'(y) = \sigma(y)$ pour tout $y \neq x$ et $\sigma'(x) = \sigma(y)[v]_{\omega''}$. Par construction et linéarité de t, $\sigma'(t) = u[v]_\omega$. D'après l'hypothèse d'induction, il existe donc une substitution σ'' telle que $\sigma''(t)$ est une valeur non-exempte de p.

Par induction sur m, il existe donc une substitution σ' telle que $\sigma'(t)$ est une valeur non-exempte de p. Comme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ et $\sigma'(t)$ est une valeur, par définition de la propriété de l'Exemption de Motif, t n'est pas exempt de p.

On considère maintenant $k > 0$ tel que pour tout terme $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ linéaire ayant strictement moins de k symboles définis, pour tout motif étendu q, si il existe une substitution telle que $\sigma'(u)$ n'est pas exempt de q, alors u n'est pas exempt de q. Comme $u = \sigma(t)$ n'est pas exempt de p, par définition de l'Exemption de Motif, il existe position $\omega \in \mathcal{P}os(u)$ avec $u_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ où $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$ telles que $u[v]_\omega$ n'est pas exempt de p.

- Si $\omega \in \mathcal{P}os(t)$, on observe que $u[v]_\omega = \sigma(t[v]_\omega)$. Donc, par hypothèse d'induction, $t[v]_\omega$ n'est pas exempt de p. De plus on rappelle que $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n) = \sum_{q \in \mathcal{Q}_u} q$ avec $\mathcal{Q}_u = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P} \text{ } t.q. \text{ } \forall j \in [1, n], u_j \text{ est exempt de } l_j\}$. Par hypothèse d'induction, on a, pour tout $i \in [1, n]$, pour tout motif étendu q si $u_i = \sigma(t_i)$ n'est pas exempt de q, alors t_i n'est pas exempt de q. Donc $\mathcal{Q}_t = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P} \text{ } t.q. \text{ } \forall j \in [1, n], t_j \text{ est exempt de } l_j\} \subseteq \mathcal{Q}_u$. Par conséquent, par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, v est exempt de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$. On conclue donc, par définition de l'Exemption de Motif que t n'est pas exempt de p.
- Sinon, il existe une position $\omega' < \omega$ telle que $t_{|\omega'} = x \in \mathcal{X}$. On note $\omega = \omega'.\omega''$, et on définit σ' la substitution telle que $\mathcal{D}om(\sigma') = \mathcal{D}om(\sigma)$ avec $\sigma'(y) = \sigma(y)$ pour toute variable $y \neq x$ et $\sigma'(x) = \sigma(y)[v]_{\omega''}$. Par construction on a donc $u[v]_\omega = \sigma'(t)$. On conclue donc, d'après l'hypothèse d'induction, que t n'est pas exempt de p.

Par induction sur k, on a donc bien démontré l'implication directe. □

1. On pourra remarquer que la preuve de l'implication directe est immédiate en utilisant la notion de sémantique introduite dans la Section suivante, avec les Propositions 3.6 et 3.7.

Et on peut également observer que les propriétés d'Exemption de Motif sont préservées pour les sous-termes, tant que ces derniers ne sont pas sous un symbole défini :

Proposition 3.4. *Étant donné un motif étendu p et un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, t est exempt de p si et seulement si $t_{|\omega}$ est exempt de p, pour toute position $\omega \in \mathcal{S}td\,(t) = \{\omega \in \mathcal{P}os(t) \mid \forall \omega' < \omega, t(\omega') \in \mathcal{C}\}$.*

Démonstration. On prouve la proposition par induction sur la forme t et k le nombre de symboles définis dans t.

Si $t = x \in \mathcal{X}$ ou $t = \varphi(t_1, \ldots, t_n)$ avec $\varphi \in \mathcal{D}^n$, la proposition est clairement vérifiée.

On considère maintenant $t = \alpha(t_1, \ldots, t_n)$ avec $\alpha \in \mathcal{C}^n$ et on prouve par induction sur k que si t est exempt de p, alors t_i est exempt de p pour tout $i \in [1, n]$.

- Si $k = 0$, alors $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. On suppose qu'il existe $i \in [1, n]$ tel que t_i n'est pas exempt de p, donc, par définition, il existe une substitution σ telle que $\sigma(t_i)$ n'est pas exempt de p, *i.e.* il existe une position $\omega \in \mathcal{P}os(\sigma(t_i))$ telle que $p \preccurlyeq \sigma(t_i)_{|\omega}$. On construit donc une substitution σ' telle que pour toute variable $x \in \mathcal{V}ar(t_i)$ $\sigma'(x) = \sigma(x)$ et pour toute variable $y_s \in \mathcal{V}ar(t) \setminus \mathcal{V}ar(t_i)$, $\sigma'(y_s) \in \mathcal{T}_s(\mathcal{C})$. On a alors, par construction, $\sigma'(t) \in \mathcal{T}(\mathcal{C})$ et $\sigma'(t)_{|i} = \sigma(t_i)$. D'où $p \preccurlyeq \sigma'(t)_{|i.\omega}$, donc $\sigma'(t)$ n'est pas exempt de p. Et on peut conclure que t n'est pas exempt de p.
- On considère maintenant $k > 0$ tel que pour tout terme $u = c(u_1, \ldots, u_n)$ ayant strictement moins de k symboles définis, si u est exempte de p alors u_i est exempt de p pour tout $i \in [1, n]$. On suppose qu'il existe $i \in [1, n]$ tel que t_i n'est pas exempt p, donc, par définition, une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \varphi_s^{!\mathcal{P}}(r_1, \ldots, r_m)$ où $\varphi \in \mathcal{D}^m$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(r_1, \ldots, r_m)$ telles que $t_i\,[v]_\omega$ n'est pas exempt de p. On note donc $u := t\,[v]_{i.\omega} = c(t_1, \ldots, t_i\,[v]_\omega, \ldots, t_n)$, et par hypothèse d'induction, u n'est pas exempt de p. Par définition de l'Exemption de Motif, on peut donc conclure que t n'est pas exempt de p.

Si t est exempt de p, on a donc, par induction sur la forme de t, que $t_{|\omega}$ est exempt de p pour toute position $\omega \in \mathcal{S}td\,(t)$. Si $t_{|\omega}$ est exempt de p pour toute position $\omega \in \mathcal{S}td\,(t)$, comme $\epsilon \in \mathcal{S}td\,(t)$, alors t est exempt de p. □

La propriété d'Exemption de Motif est donc bien préservée par substitution, ainsi que pour tous les sous-termes ne se trouvant pas sous un symbole défini. En effet, on peut considérer ces positions comme "stables", dans le sens où les symboles à ces positions resteront inchangés par la réduction induite d'un CBTRS : on souhaite que ces sous-termes conservent les propriétés d'Exemption de Motif du terme, pour qu'elles soient également conservées par la relation de réécriture elle-même.

On remarque cependant que la préservation de l'Exemption de Motif par substitution n'est vérifiée que pour les termes linéaires. On étudiera de façon plus approfondie le cas des termes non-linéaires dans la Section 3.4 et le Chapitre 5.

Exemple 3.7. *On considère dans cet exemple les termes de l'algèbre définie par la signature $\Sigma_{nl} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :*

$$\begin{array}{rcl} \mathsf{S1} & := & c(\mathsf{S2}, \mathsf{S2}) \\ & \mid & d(\mathsf{S1}) \end{array} \qquad\qquad \begin{array}{rcl} \mathsf{S2} & := & a \\ & \mid & b \end{array}$$

avec les symboles définis $\mathcal{D} = \{f^{!\mathcal{P}_f} : \mathsf{S1} \mapsto \mathsf{S1}, g^{!\mathcal{P}_g} : \mathsf{S1} \mapsto \mathsf{S2}\}$, où $\mathcal{P}_f = \{\bot \mapsto c(a, b)\}$ et $\mathcal{P}_g = \emptyset$.

On peut observer que $c(x, a)$ est exempt de $c(a, b)$ et ainsi que pour toute substitution σ, $\sigma(c(x, a)) = c(\sigma(x), a)$ est exempt de $c(a, b)$. En revanche, bien que $c(x, x)$ soit également exempt $c(a, b)$, en considérant $\sigma = \{x \mapsto g(z)\}$, on a $\sigma(c(x, x)) = c(g(z), g(z))$ qui n'est pas exempt de $c(a, b)$, puisqu'en remplaçant le premier $g(z)$ par a, comme décrit dans la Définition 3.3, on obtient $c(a, g(z))$ qui n'est pas exempt de $c(a, b)$.

On peut également observer que $d(f(c(a, b)))$ est exempt de $c(a, b)$ même si le sous-terme en argument du symbole défini f n'est clairement pas exempt de $c(a, b)$.

Comme la notion d'Exemption de Motif considère, et définit, une sur-approximation des formes normales obtenues par réécriture, il est nécessaire de prouver que le CBTRS considéré induit bien une relation de réécriture conservant les propriétés d'Exemption de Motif. En effet, les profils choisis supposent donc que les formes normales éventuelles seront compatibles avec cette sur-approximation, et cette supposition reste maintenant à vérifier.

Pour se faire, il est non seulement nécessaire de pouvoir vérifier des propriétés d'Exemption de Motif de manière systématique, mais aussi de pouvoir examiner efficacement la sur-approximation considérée. Dans les deux cas, on voit qu'il est potentiellement nécessaire de considérer un nombre infini de valeurs de l'algèbre. On propose donc d'utiliser une extension de la notion de sémantique close introduite dans [CM19], où elle est déjà utilisée comme une construction finie permettant de représenter une nombre potentiellement infini de valeurs.

3.2 Motifs étendus et Sémantique

En partant de l'observation qu'un terme de l'algèbre peut être interprété comme l'ensemble de ses instances, une notion de sémantique close est introduite dans [CM19] pour représenter, à partir d'un motif linéaire, l'ensemble potentiellement infini des valeurs filtrées par ce motif. Cette approche est originellement utilisée pour traduire certaines opérations ensemblistes (notamment l'union et la différence) par des motifs étendus (via les opérateurs binaires de disjonction + et de complément \).

Dans le contexte d'une algèbre de termes contenant des symboles définis, non seulement cette notion de sémantique close peut être généralisée pour représenter la sur-approximation de formes normales potentielles considérée dans la Définition 3.3 de la propriété d'Exemption de Motif, mais elle permettra également d'exprimer et de vérifier des conditions nécessaires et suffisantes à cette même Exemption de Motif.

3.2.1 Sémantique de Motif généralisée

Étant donné un terme constructeur $p \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, la sémantique close de p [CM19], notée $[\![p]\!]$, est l'ensemble des instances constructeures closes de p : $[\![p]\!] = \{\sigma(p) \mid \sigma(p) \in \mathcal{T}(\mathcal{C})\}$. De plus, cette notion établit une équivalence avec le filtrage par motif : $v \in [\![p]\!]$ si et seulement si $p \preccurlyeq v$. Il est ainsi possible d'utiliser cette sémantique close pour comparer la forme en tête de termes constructeurs : si $[\![p]\!] \cap [\![q]\!] = \emptyset$, alors on a, pour toute substitution σ, $\sigma(q) \notin [\![p]\!]$, *i.e.* $p \not\preccurlyeq \sigma(q)$.

Etant donné un terme constructeur $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on peut donc comparer sa sémantique à celle d'un motif linéaire p pour déterminer si t est exempt de p :

Exemple 3.8. *On considère l'algèbre de listes introduite dans l'Exemple 3.5. On peut facilement vérifier que $[\![cons(e, nil)]\!] \cap [\![cons(lst(nil), l)]\!] = \{cons(lst(nil), nil)\}$, et donc déduire que $cons(e, nil)$ n'est pas exempt de $cons(lst(nil), l)$, et inversement que $cons(lst(nil), l)$ n'est pas exempt de $cons(e, nil)$.*

De même, on a $[\![cons(int(i), nil)]\!] \cap [\![cons(lst(l_1), l_2)]\!] = \emptyset$ *et* $[\![nil]\!] \cap [\![cons(lst(l_1), l_2)]\!]$, *et comme la sorte* Int *ne peut contenir que les constructeurs s et z, on sait que* $int(i)$ *est exempt de* $cons(lst(l_1), l_2)$ *et on peut conclure que* $cons(int(i), nil)$ *est exempt de* $cons(lst(l_1), l_2)$.

On souhaite donc généraliser cette approche à l'ensemble des termes de l'algèbre, et ainsi que la sémantique d'un terme contenant un symbole défini corresponde à la sur-approximation de la forme normale du terme, comme définie pour la propriété d'Exemption de Motif (Définition 3.3) :

Définition 3.4 (Sémantique généralisée)**.** *Soient* $u \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ *et* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{C}, \mathcal{X})$, *on définit la sémantique close de u, respectivement t, notés* $[\![u]\!]$ *et* $[\![t]\!]$, *par :*

- $[\![u]\!] = \{\sigma(u) \mid \sigma(u) \in \mathcal{T}(\mathcal{C})\}$;
- $[\![t]\!] = \bigcup_\omega \bigcup_v [\![t\,[v]_\omega]\!]$, *avec* $\omega \in \mathcal{P}os(t)$ *telle que* $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ *avec* $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, *et* $v \in \mathcal{T}_s(\mathcal{C})$ *exempte de* $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$.

où $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$ *est le* motif annotant *de* $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$, *défini par* $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n) = \sum_{q \in \mathcal{Q}} q$ *avec* $\mathcal{Q} = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ *t.q.* $\forall i \in [1, n], t_i$ *est exempt de* $l_i\}$.

Cette définition étant une généralisation de la sémantique close aux termes contenant des symboles définis, la définition est inchangée pour les termes constructeurs. Aussi, par la suite, la notion de sémantique close d'un terme référera à cette généralisation. La sémantique close d'un terme contenant des symboles définis est l'union des sémantiques closes des termes obtenues par sur-approximation des réductions possibles des sous-termes ayant un symbole défini comme symbole de tête. Comme expliqué dans la Section précédente, cette sur-approximation est définie et contrôlée par les profils utilisés en annotation du symbole défini, et est donc identique à celle utilisée dans la Définition 3.3 de l'Exemption de Motif. Ainsi, plus les profils décrivent le comportement de la fonction considérée de façon précise, plus cette sur-approximation sera précise.

On peut remarquer que la sémantique d'une variable de sorte donnée est l'ensemble des valeurs de l'algèbre de la même sorte, et, dans le cas d'un terme linéaire, on peut ainsi donner une définition récursive de sa sémantique :

Proposition 3.5.

- *Soit une variable* $x_s \in \mathcal{X}$ *avec* $s \in \mathcal{S}$, *on a :*

$$[\![x_s]\!] \;=\; \mathcal{T}_s(\mathcal{C}) \;=\; \bigcup_{c \in \mathcal{C}_s} [\![c(z_{1s_1}, \ldots, z_{ns_n})]\!] \quad \text{avec } \mathcal{D}om(c) = s_1 * \cdots * s_n$$

- *Soit un terme linéaire* $c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *avec* $c \in \mathcal{C}^n$, *on a :*

$$[\![c(t_1, \ldots, t_n)]\!] \;=\; \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$$

- *Soit un terme* $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *avec* $\varphi \in \mathcal{D}^n$, *on a :*

$$[\![\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)]\!] \;=\; \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \text{ exempt de } \triangleright^{\mathcal{P}}(t_1, \ldots, t_n)\}$$

Démonstration.

- Soit une variable $x_s \in \mathcal{X}$ avec $s \in \mathcal{S}$, on a : $v \in [\![x_s]\!]$ si et seulement s'il existe une substitution sortée σ telle que $\sigma(x_s) = v$. Si $v \in \mathcal{T}_s(\mathcal{C})$, alors on peut définir $\sigma = \{x_s \mapsto v\}$, et sinon, toute substitution associant x_s à v est mal sortée, d'où la première égalité.

Pour la deuxième égalité, il est clair qu'étant donné un constructeur $c : s_1 * \cdots * s_n \mapsto s$, toute valeur $v \in [\![c(z_{1_{s_1}}, \ldots, z_{n_{s_n}})]\!]$ est une valeur de sorte s, *i.e.* $v \in \mathcal{T}_s(\mathcal{C})$. Et inversement, si $v \in \mathcal{T}_s(\mathcal{C})$, alors il existe $c \in \mathcal{C}_s$ tel que $v = c(v_1, \ldots, v_n)$, avec $arity\,(c) = n$. En notant $s_1 * \cdots * s_n$ le domaine de c, on peut donc construire $\sigma = \{z_{1_{s_1}} \mapsto v_1, \ldots, z_{n_{s_n}} \mapsto v_n\}$ pour avoir $v = \sigma(c(z_{1_{s_1}}, \ldots, z_{n_{s_n}}))$, d'où $v \in [\![c(z_{1_{s_1}}, \ldots, z_{n_{s_n}})]\!]$.

- Soit un terme linéaire $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}$, on procède par induction sur k, le nombre de symboles définis dans t. Si $k = 0$, alors $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, et comme t linéaire, on a $v \in [\![t]\!]$, *i.e.* $c(t_1, \ldots, t_n) \nprec v$ si et seulement $v = c(v_1, \ldots, v_n)$, avec $t_i \nprec v_i, \forall i \in [1, n]$, *i.e.* $v_i \in [\![t_i]\!], \forall i \in [1, n]$, donc t vérifie la propriété. On suppose maintenant $k > 0$ tel que $\forall u = c(u_1, \ldots, u_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ayant moins de k symboles définis, $[\![u]\!] = \{c(w_1, \ldots, w_n) \mid (w_1, \ldots, w_n) \in [\![u_1]\!] \times \cdots \times [\![u_n]\!]\}$.
  On considère $v \in [\![t]\!]$; par définition, il existe une position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(p_1, \ldots, p_m)$ avec $\varphi \in \mathcal{D}$, et une valeur $w \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(p_1, \ldots, p_m)$ telles que $v \in [\![t\,[w]_\omega]\!]$. Comme $t(\epsilon) = c \in \mathcal{C}$, $\omega \neq \epsilon$ et on peut donc noter $\omega = i.\omega'$. Ainsi, par induction, $v = c(v_1, \ldots, v_n)$ avec $v_j \in [\![t_j]\!], \forall j \neq i$ et $v_i \in [\![t_i\,[w]_{\omega'}]\!]$. De plus, par définition, on a $[\![t_i\,[w]_{\omega'}]\!] \subseteq [\![t_i]\!]$, d'où $v \in \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$.
  On considère maintenant $v \in \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$, donc $v = c(v_1, \ldots, v_n)$ avec $v_i \in [\![t_i]\!], \forall i \in [1, n]$. Comme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et $t(\epsilon) = c \in \mathcal{C}$, il existe $i \in [1, n]$ tel que $t_i \notin \mathcal{T}(\mathcal{C}, \mathcal{X})$. Donc il existe une position $\omega \in \mathcal{P}os(t_i)$ telle que $t_{i|\omega} = \varphi_s^{!\mathcal{P}}(p_1, \ldots, p_m)$ avec $\varphi \in \mathcal{D}^m$, et une valeur $w \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(p_1, \ldots, p_m)$ telles que $v_i \in [\![t_i\,[w]_\omega]\!]$. Ainsi, par induction, on a $v \in [\![t\,[w]_{i.\omega}]\!]$, et comme, par définition, $[\![t\,[w]_{i.\omega}]\!] \subseteq [\![t]\!]$, $v \in [\![t]\!]$.
  Ainsi, par induction, l'égalité est vérifiée pour tout terme linéaire de la forme $c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}^n$. [1]
- Soit un terme $t = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $\varphi \in \mathcal{D}^n$. Par définition de la sémantique close, on a pour toute valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$, $[\![t\,[v]_\epsilon]\!] \subseteq [\![t]\!]$, *i.e.* $v \in [\![t]\!]$.
  On prouve maintenant, par induction sur k le nombre de symboles définis dans t que pour tout $v \in [\![t]\!]$, v est une valeur de sorte s exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$. Si $k = 1$, ϵ est la seule position de t dont le symbole est un symbole défini, donc la définition de la sémantique close garantit que v est bien une valeur de sorte s exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$. On suppose maintenant $k > 0$ tel que $\forall u = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ayant moins de k symboles définis, si $v \in [\![u]\!]$ alors v est une valeur de sorte s exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$.
  Soit $v \in [\![t]\!]$, par définition, il existe une position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \psi_{s'}^{!\mathcal{P}'}(p_1, \ldots, p_m)$ avec $\psi \in \mathcal{D}^m$, et une valeur $w \in \mathcal{T}_{s'}(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}'}(p_1, \ldots, p_m)$ telles que $v \in [\![t\,[w]_\omega]\!]$. Si $\omega = \epsilon$, alors v est une valeur de sorte s exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$. Sinon, on note $\omega = i.\omega'$ et $\varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n) = t\,[w]_\omega$, et par induction on sait que v est également une valeur de sorte s exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$. On rappelle que $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n) = \sum_{q \in \mathcal{Q}_u} q$ avec $\mathcal{Q}_u = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}\ t.q.\ \forall j \in [1, n], u_j$ est exempt de $l_j\}$. De plus, par définition de l'Exemption de Motif, pour tout profil $l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$, si t_i est exempt de l_i, u_i l'est aussi, d'où $\mathcal{Q}_t = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}\ t.q.\ \forall j \in [1, n], t_j$ est exempt de $l_j\} \subseteq \mathcal{Q}_u$. Par conséquent, par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, v est exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$.
  Ainsi, par induction, l'égalité est vérifiée pour tout terme de la forme $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $\varphi \in \mathcal{D}^n$.

□

1. Dans le cas général, on montre également l'inclusion directe avec le Lemme A.1 prouvé en Annexe A

De plus, comme pour la propriété d'Exemption de Motif, la sémantique close est conservée par substitution (pour les termes linéaires) :

Proposition 3.6. *Soient un terme linéaire $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et une substitution σ, on a $[\![\sigma(t)]\!] \subseteq [\![t]\!]$.*

Démonstration. Soient $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et une substitution σ, on procède par induction sur la forme de t :

- Si $t = x_s$, on a $\sigma(x_s) : s$, donc on peut remarquer que par définition de la sémantique close, pour tout $v \in [\![\sigma(x_s)]\!]$, v est de sorte s, d'où $[\![\sigma(x_s)]\!] \subseteq \mathcal{T}_s(\mathcal{C}) = [\![x_s]\!]$.
- Si $t = c(t_1, \ldots, t_n)$ avec $c \in \mathcal{C}^n$ et $t_1, \ldots, t_n$ des termes linéaires tels que $\forall i, [\![\sigma(t_i)]\!] \subseteq [\![t_i]\!]$, on a donc $[\![\sigma(t_1)]\!] \times \cdots \times [\![\sigma(t_n)]\!] \subseteq [\![t_1]\!] \times \cdots \times [\![t_n]\!]$. Par conséquent, avec Proposition 3.5, on a $[\![\sigma(t)]\!] \subseteq [\![t]\!]$.
- Si $t = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi \in \mathcal{D}^n$ et $t_1, \ldots, t_n$ des termes linéaires tels que $\forall i, [\![\sigma(t_i)]\!] \subseteq [\![t_i]\!]$, on a, d'après Proposition 3.5, $[\![t]\!] = \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \text{ exempt de } \triangleright^{\mathcal{P}}(t_1, \ldots, t_n)\}$ et $[\![\sigma(t)]\!] = \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \text{ exempt de } \triangleright^{\mathcal{P}}(\sigma(t_1), \ldots, \sigma(t_n))\}$. On rappelle que $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n) = \sum_{q \in \mathcal{Q}} q$ avec $\mathcal{Q} = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P} \text{ t.q. } \forall j \in [1, n], u_j \text{ est exempt de } l_j\}$. De plus, pour tout profil $l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$, pour tout $i \in [1, n]$, comme $[\![\sigma(t_i)]\!] \subseteq [\![t_i]\!]$, Proposition 3.7 garantit que si t_i est exempt de l_i, alors $\sigma(t_i)$ l'est aussi, d'où $\{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P} \text{ t.q. } \forall i \in [1, n], t_i \text{ est exempt de } l_i\} \subseteq \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P} \text{ t.q. } \forall i \in [1, n], \sigma(t_i) \text{ est exempt de } l_i\}$. Donc, par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, $\{v \mid v \in \mathcal{T}_s(\mathcal{C}) \text{ exempt de } \triangleright^{\mathcal{P}}(\sigma(t_1), \ldots, \sigma(t_n))\} \subseteq \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \text{ exempt de } \triangleright^{\mathcal{P}}(t_1, \ldots, t_n)\}$, donc $[\![\sigma(t)]\!] \subseteq [\![t]\!]$.

  Par induction, l'inclusion est donc vérifiée pour tout terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ linéaire. □

Comme pour les propriétés d'Exemption de Motif (Propriété 3.3), on remarque que la sémantique close n'est préservée par substitution que pour les termes linéaires (on peut le voir simplement en reprennant l'Exemple 3.7, en considérant la sémantique des différents termes). On étudiera de façon plus approfondie le cas des termes non-linéaires dans la Section 3.4 et le Chapitre 5.

La sémantique close d'un terme étant donc équivalente à la sur-approximation considérée dans la Définition 3.3 de l'Exemption de Motif, vérifier une propriété d'Exemption de Motif d'un terme de l'algèbre revient donc à la vérifier pour toutes les valeurs de sa sémantique :

Proposition 3.7. *Soit $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et un motif étendu p, t est exempt de p si et seulement si, $\forall v \in [\![t]\!]$, v est exempt de p.*

Démonstration. Pour $t \in \mathcal{T}(\mathcal{C})$, $[\![t]\!] = \{t\}$, donc la relation est clairement vérifiée.

Pour $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{C})$, t est exempt de p si et seulement si, pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{C})$, $\sigma(t)$ est exempt de p. Donc, par définition de la sémantique close, t est exempt de p si et seulement si $\forall v \in [\![t]\!]$, v est exempt de p.

Pour $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on prouve la relation par induction sur k, le nombre de symboles définis dans t. Si $k = 0$, alors $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, et donc t vérifie la propriété. On suppose maintenant $k > 0$ tel que $\forall u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ayant moins de k symboles définis, u est exempt de p si et seulement si $\forall v \in [\![u]\!]$, v est exempt de p. Par définition, $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{C}, \mathcal{X})$ est exempt de p si et seulement si, pour toute position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et pour toute valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1, \ldots, t_n)$, le terme $t[v]_\omega$ est exempt de p. De plus, étant donné une telle position ω et une telle valeur v, le nombre de symboles définis dans $t[v]_\omega$ est strictement inférieur à k, donc $t[v]_\omega$ est exempt de p si et seulement $\forall w \in [\![t[v]_\omega]\!]$, w est exempt de p. Donc,

par définition de la sémantique close, on a bien t exempt de p si et seulement si $\forall w \in [\![t]\!]$, w est exempt de p.

Ainsi, par induction, la relation est vérifiée pour tout $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. □

Cette relation fournit une première condition nécessaire et suffisante à la propriété d'Exemption de Motif, permettant de mettre en place une méthode générale de vérification de cette dernière. En effet, la notion d'Exemption de Motif pour les valeurs (Définition 3.1), est simple et facilement vérifiable : il suffit de tester le filtrage pour tout sous-terme de la valeur. On peut donc en premier approche établir qu'un terme t est exempt d'un motif p en vérifiant qu'il n'est pas possible d'identifier une valeur non-exempte de p dans la sémantique $[\![t]\!]$:

Exemple 3.9. *On considère l'algèbre de termes introduite dans l'Exemple 3.5. On note* $p_{flat} = cons(lst(l_1), l_2)$*, et on rappelle que les symboles définis sont annotés* $flatten^{!\mathcal{P}_1}$ *et* $concat^{!\mathcal{P}_2}$ *avec* $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ *et* $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$.

- *On remarque* $cons(lst(nil), nil) \in [\![cons(e, l)]\!]$*, et* $cons(lst(nil), nil)$ *n'est clairement pas exempt de* p_{flat} *(puisque filtré par* p_{flat}*) donc* $cons(e, l)$ *n'est pas exempt de* p_{flat}.
- *De même, on peut voir que* $cons(int(z), cons(lst(nil), nil)) \in [\![cons(int(i), l)]\!]$ *n'est pas exempt de* p_{flat}*, puisque* p_{flat} *filtre le sous-terme* $cons(lst(nil), nil)$*. Donc* $cons(int(i), l)$ *n'est pas exempt de* p_{flat}.
- *On peut facilement montrer que, pour tout* $v \in [\![cons(int(i), nil)]\!]$*,* v *est exempt de* p_{flat}*, puisque* v *est forcément un terme de la forme* $cons(int(n), nil)$ *avec* n *une valeur de sorte* Int*. Ainsi* v *n'est pas filtré par* p_{flat} *et ne contient qu'un seul sous-terme de sorte* List*,* nil*, qui n'est également pas filtré par* p_{flat}*. Le sous-terme* $int(n)$ *étant de sorte* Expr*, la valeur* n*, et ses (éventuels) sous-termes, de sorte* Int*, ils ne peuvent également pas être filtrés par* p_{flat}*. On peut donc en déduire que* $cons(int(i), nil)$ *est exempt de* p_{flat}.
- *Étant donné que* l *est exempt de* $\bot$*, pour tout* $v \in [\![flatten^{!\mathcal{P}_1}(l)]\!]$*, par définition,* v *est exempt de* p_{flat}*. Par conséquent* $flatten^{!\mathcal{P}_1}(l)$ *est exempt de* p_{flat}.
- *On peut en déduire que, comme* $flatten^{!\mathcal{P}_1}(l_1)$ *et* $flatten^{!\mathcal{P}_1}(l_2)$ *sont exempts de* p_{flat} *et que l'annotation* $\mathcal{P}_2$ *contient le profil* $p_{flat} * p_{flat} \mapsto p_{flat}$*, par définition de la sémantique, pour tout* $v \in [\![concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))]\!]$*,* v *est exempt de* p_{flat}*. Par conséquent,* $concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))$ *est exempt de* p_{flat}.
- *On peut voir que* $cons(lst(nil), nil) \in [\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!]$*, et* $cons(lst(nil), nil)$ *n'est clairement pas exempt de* p_{flat} *(puisque filtré par* p_{flat}*) donc* $cons(e, flatten^{!\mathcal{P}_1}(l))$ *n'est pas exempt de* p_{flat}.
- *Enfin, on peut montrer que, pour tout* $v \in [\![cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))]\!]$*,* v *est exempt de* p_{flat}*, puisque, par définition de la sémantique,* v *est forcément de la forme* $cons(int(n), w)$ *avec* n *une valeur de sorte* Int *et* w *une valeur de sorte* List *exempte de* p_{flat} *issue de la sur-approximation de* $flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))$*. Ainsi* p_{flat} *ne filtre pas* v*, et tout autre sous-terme de* v *de sorte* List *est soit* w*, soit un de ses (potentiels) sous-termes. Ce dernier étant exempt de* p_{flat}*, ni* w*, ni ses sous-termes ne sont filtrés par* p_{flat}*. Donc* $cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))$ *est exempt de* p_{flat}.

Bien que cette approche permette de proposer une méthode générale pour la vérification de propriétés d'Exemption de Motif, il est toujours nécessaire ici de considérer un nombre potentiellement infini de valeurs contenues dans la sémantique du terme considéré. Ainsi plutôt que de chercher à considérer toutes ces valeurs individuellement, on propose donc d'étendre, par la suite, la notion de sémantique close aux motifs étendus (comme cela avait également été fait

dans [CM19]), comme l'ensemble des termes filtrés par le motif considéré, ce qui permettra de reformuler la condition nécessaire et suffisante, donnée par la Proposition 3.7.

3.2.2 Variables annotées et motifs étendus

En partant de l'observation que la sémantique d'un terme ayant un symbole défini comme symbole de tête est l'ensemble de valeurs de même sorte et exemptes du *motif annotant* issu de l'annotation du symbole et de ses sous-termes (Proposition 3.5), on souhaite pouvoir exprimer une telle sémantique directement à partir du *motif annotant* et de la sorte considérée. On peut également remarquer que quand le *motif annotant* est $\bot$, comme toute valeur est exempte de $\bot$, cette sémantique est équivalente à la sémantique d'une variable de même sorte.

On introduit donc une notion de variable annotée : toute variable peut être annotée par un motif additif linéaire pour indiquer une propriété d'Exemption de Motif nécessaire au filtrage.

Définition 3.5 (Variables annotées)**.** *On considère un ensemble de variables annotées, noté $\mathcal{X}^a$, tel que toute variable $x_s^{-p} \in \mathcal{X}^a$ est de sorte s et annotée par le motif additif linéaire $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$.*

Une variable x_s^{-p} induit une relation de filtrage :

$$x_s^{-p} \nprec v \iff v : s \text{ exempte de } p$$

On remarque qu'on a donc bien qu'une variable $x_s \in \mathcal{X}$ est équivalente à une variable $x_s^{-\bot} \in \mathcal{X}^a$, puisque tout terme est exempt de $\bot$. On pourra donc utiliser et noter de façon indifférenciée x_s et $x_s^{-\bot}$.

En considérant l'annotation des variables, on étend la notion de sémantique close aux termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$:

Définition 3.6. *Soit un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, on définit la sémantique close de t par $[\![t]\!] = \{\sigma(t) \in \mathcal{T}(\mathcal{C}) \mid \forall x_s^{-p} \in \mathcal{V}ar(t), \sigma(x_s^{-p}) \text{ est exempt de } p\}$.*

On considère donc par la suite des motifs étendus incluant des variables annotées en plus des opérateurs de disjonction et de complément. On introduit également un opérateur de conjonction de motifs et un opérateur d'aliasing permettant de restreindre le filtrage d'une variable.

Définition 3.7 (Motif étendu)**.** *Étant donné un ensemble de variables annotées $\mathcal{X}^a$ et une signature $\Sigma = (\mathcal{S}, \mathcal{D} \uplus \mathcal{C})$, l'ensemble $\mathcal{P}(\mathcal{C}, \mathcal{X}^a)$ des* motifs étendus *est défini comme l'ensemble des termes de la forme :*

$$p, q \quad := \quad x \mid c(q_1, \ldots, q_n) \mid p_1 + p_2 \mid p_1 \setminus p_2 \mid p_1 \times p_2 \mid y \mathbin{@} p_2 \mid \bot$$

avec $x \in \mathcal{X}_s^a, y \in \mathcal{X}_s, p, p_1, p_2 : s \in \mathcal{S}, c : s_1 * \cdots * s_n \mapsto s \in \mathcal{C} \cup \mathcal{N}$ et $\forall i \in [1, n], q_i : s_i$.

Dans ce contexte, on appelle :

- réguliers *les motifs dont toutes les variables sont annotées $-\bot$, i.e. les motifs de $\mathcal{P}(\mathcal{C}, \mathcal{X})$;*
- symboliques *les motifs étant des termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$;*
- additifs *les motifs ne contenant ni $\times$ ni $\setminus$;*
- quasi-additifs *les motifs ne contenant pas $\times$, et contenant seulement $\setminus$ avec le motif à gauche étant une variable et le motif à droite étant régulier et additif ;*
- quasi-symboliques *les motifs quasi-additifs ne contenant pas $\bot$ et ne contenant $+$ seulement à droite d'un $\setminus$.*

En plus des opérateurs de disjonction + et de complément \ qui gardent leurs définitions respectives, l'opérateur de conjonction × permet d'exprimer la conjonction des propriétés de filtrage : $p_1 \times p_2$ filtre une valeur si et seulement si cette dernière est filtrée par p_1 et p_2. L'opérateur d'aliasing @ se comporte comme une conjonction, mais permet plus précisément d'expliciter une contrainte d'instanciation d'une variable : $x \mathbin{@} p$ restreint le filtrage de la variable x aux valeurs filtrées par p.

La notion de filtrage se généralise naturellement pour les motifs étendus, dans le cas de motifs linéaires. On définit donc la notion de linéarité pour les motifs étendus via une notion de méta-variables, représentant les variables restreintes par le filtrage :

Définition 3.8 (Méta-variables)**.** *Étant donné un motif étendu $p \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$, l'ensemble $\mathcal{MV}(p)$ des* méta-variables *du motif p est défini par :*

$$\begin{aligned}
\mathcal{MV}(x) &= \{x\}, \textit{pour } x \in \mathcal{X}^a \\
\mathcal{MV}(c(p_1, \ldots, p_n)) &= \mathcal{MV}(p_1) \cup \ldots \cup \mathcal{MV}(p_n), \textit{pour tout } c \in \mathcal{C}^n \\
\mathcal{MV}(p_1 + p_2) &= \mathcal{MV}(p_1) \cup \mathcal{MV}(p_2) \\
\mathcal{MV}(p_1 \times p_2) &= \mathcal{MV}(p_1) \cup \mathcal{MV}(p_2) \\
\mathcal{MV}(p_1 \setminus p_2) &= \mathcal{MV}(p_1) \\
\mathcal{MV}(x \mathbin{@} p) &= \mathcal{MV}(x \times p) \\
\mathcal{MV}(\bot) &= \emptyset
\end{aligned}$$

On peut ainsi définir la notion de linéarité des motifs étendus par indépendance des méta-variables :

Définition 3.9 (Linéarité)**.** *Un motif étendu de la forme :*

- *$c(p_1, \ldots, p_n)$ est linéaire si et seulement si chaque p_i, $i \in [1, n]$, est linéaire, et $\forall 1 \leq i < j \leq n$, $\mathcal{MV}(p_i) \cap \mathcal{MV}(p_j) = \emptyset$;*
- *$p_1 + p_2$ est linéaire si et seulement si p_1 et p_2 sont linéaires ;*
- *$p_1 \setminus p_2$ est linéaire si et seulement si p_1 et p_2 sont linéaires ;*
- *$p_1 \times p_2$ est linéaire si et seulement si p_1 et p_2 sont linéaires et $\mathcal{MV}(p_1) \cap \mathcal{MV}(p_2) = \emptyset$;*
- *$x \mathbin{@} p$ est linéaire si et seulement si $x \times p$ est linéaire.*

Intuitivement, pour que $c(p_1, \ldots, p_n)$ soit linéaire, les motifs $p_i, i \in [1, n]$ doivent être linéaires et deux à deux indépendants, tandis que pour $p_1 + p_2$, resp. $p_1 \setminus p_2$, p_1 et p_2 représentent des alternatives indépendantes, donc leurs variables sont également indépendantes. Pour $p_1 \times p_2$, p_1 et p_2 ne sont plus des alternatives indépendantes puisque les deux doivent vérifier la relation de filtrage, donc ils doivent être linéaires et indépendants pour que $p_1 \times p_2$ soit linéaire. Enfin, $x \mathbin{@} p$ restreint le filtrage de la variable x par le motif p, et donc se comporte comme une conjonction $x \times p$.

Exemple 3.10. *On considère des motifs de l'algèbre de listes introduite dans l'Exemple 3.5 :*

- *Le motif $cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)) + lst(nil), l \mathbin{@} lst(l))$ est quasi-additif et linéaire mais pas quasi-symbolique puisqu'on a un + directement sous le symbole cons ;*
- *Le motif $cons(lst(l \mathbin{@} (y_{\mathsf{List}}^{-p_{flat}} \setminus (lst(nil) + int(i)))), l \setminus nil)$ est quasi-symbolique mais non-linéaire puisque la variable l apparaît dans les sous-termes du symbole cons ;*
- *Le motif $cons(e \mathbin{@} (lst(cons(e, nil))), z_{\mathsf{List}}^{-p_{flat}} \setminus cons(int(i), l))$ est quasi-symbolique mais non-linéaire puisque la variable e apparaît à droite de son alias dans v.*

Par la suite, on ne considère que des motifs étendus conjonctions-linéaires, *i.e.* tels que pour toute conjonction $p_1 \times p_2$, on a $\mathcal{MV}(p_1) \cap \mathcal{MV}(p_2) = \emptyset$, et pour tout alias $x \mathbin{@} p$, on a $x \notin \mathcal{MV}(p)$. De plus, les termes à droite d'une conjonction seront toujours linéaires.

On peut ainsi donner une définition opérationnelle du filtrage par motif étendu linéaire [1] :

Définition 3.10 (Filtrage étendu)**.** *Soient un motif additif régulier et linéaire $q \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$ et une valeur $v \in \mathcal{T}(\mathcal{C})$, on note $\preccurlyeq$ la relation de filtrage par motif étendu, définie telle que :*

$$\begin{array}{rcll} x_s^{-q} & \preccurlyeq & v & \iff v : s \text{ exempte de } q \quad \text{pour } x_s^{-q} \in \mathcal{X}^a \\ c(p_1, \ldots, p_n) & \preccurlyeq & c(v_1, \ldots, v_n) & \iff \wedge_{i=1}^n p_i \preccurlyeq v_i \quad \text{pour } c \in \mathcal{C}^n \\ p_1 + p_2 & \preccurlyeq & v & \iff p_1 \preccurlyeq v \ \vee \ p_2 \preccurlyeq v \\ p_1 \times p_2 & \preccurlyeq & v & \iff p_1 \preccurlyeq v \ \wedge \ p_2 \preccurlyeq v \\ p_1 \setminus p_2 & \preccurlyeq & v & \iff p_1 \preccurlyeq v \ \wedge \ p_2 \not\preccurlyeq v \\ x \mathbin{@} p & \preccurlyeq & v & \iff x \times p \preccurlyeq v \end{array}$$

On généralise la sémantique aux motifs étendus, de façon à garder la corrélation entre la sémantique close et le filtrage par motif :

Définition 3.11 (Sémantique close des motifs étendus)**.** *Soit un motif linéaire étendu $p \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$, on définit la sémantique close de p, notée $[\![p]\!]$, comme l'ensemble des valeurs filtrées par p :*

$$[\![p]\!] \quad = \quad \{v \in \mathcal{T}(\mathcal{C}) \mid p \preccurlyeq v\}$$

On peut remarquer que, pour un terme constructeur de $\mathcal{T}(\mathcal{C}, \mathcal{X})$, cette définition est bien équivalente à la Définition 3.4. De plus, on peut donner une définition récursive de cette sémantique en fonction de la forme du motif étendu :

Proposition 3.8.

- *Soit une variable $x_s^{-p} \in \mathcal{X}^a$ avec $s \in \mathcal{S}$ et $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$ additif linéaire, on a :*

$$[\![x_s^{-p}]\!] \quad = \quad \bigcup_{c \in \mathcal{C}_s} [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p}) \setminus p]\!] \quad \text{avec } \mathcal{D}om(c) = s_1 * \cdots * s_n$$

- *Soit un motif linéaire $c(p_1, \ldots, p_n) \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$, on a :*

$$[\![c(p_1, \ldots, p_n)]\!] \quad = \quad \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![p_1]\!] \times \cdots \times [\![p_n]\!]\}$$

- *Soient des motifs linéaires p_1 et p_2, on a :*

$$\begin{array}{rcl} [\![p_1 + p_2]\!] & = & [\![p_1]\!] \cup [\![p_2]\!] \\ [\![p_1 \setminus p_2]\!] & = & [\![p_1]\!] \setminus [\![p_2]\!] \end{array}$$

- *Soient des motifs linéaires p_1 et p_2, tels que $\mathcal{MV}(p_1) \cap \mathcal{MV}(p_2) = \emptyset$, on a :*

$$[\![p_1 \times p_2]\!] \quad = \quad [\![p_1]\!] \cap [\![p_2]\!]$$

- *Soient une variable $x \in \mathcal{X}_s$ et un motif linéaire $p : s$ tels que $x \notin \mathcal{MV}(p)$, on a :*

$$[\![x \mathbin{@} p]\!] \quad = \quad [\![p]\!]$$

1. Une définition applicable aux motifs non-linéaires sera donnée dans le Chapitre 5

Démonstration.

- Soit une variable $x_s^{-p} \in \mathcal{X}^a$ avec $s \in \mathcal{S}$ et $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$, on montre que :

$$[\![x_s^{-p}]\!] \quad = \quad \bigcup_{c \in \mathcal{C}_s} [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p}) \setminus p]\!] \quad \text{avec } \mathcal{D}om(c) = s_1 * \cdots * s_n$$

  On considère $v \in \bigcup_{c \in \mathcal{C}_s} [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p}) \setminus p]\!]$, *i.e.* il existe $c \in \mathcal{C}_s$, avec $\mathcal{D}om(c) = s_1 * \cdots * s_n$, tel que $v \in [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p})]\!]$ et $v \notin [\![p]\!]$. Par linéarité, on sait, d'après Proposition 3.5, qu'il existe $(v_1, \ldots, v_n) \in [\![z_1 s_1^{-p}]\!] \times \cdots \times [\![z_n s_n^{-p}]\!]$ tel que $v = c(v_1, \ldots, v_n)$, et $p \not\preccurlyeq v$. Par exemption de p pour $v_1, \ldots, v_n$, on a donc, pour toute position $\omega \in \mathcal{P}os(v)$, $p \not\preccurlyeq v_{|\omega}$, donc v est exempt de p, d'où $v \in [\![x_s^{-p}]\!]$.

  On considère $v \in [\![x_s^{-p}]\!]$, donc $v : s$ et v est exempt de p. Donc il existe $c \in \mathcal{C}_s$ avec $\mathcal{D}om(c) = s_1 * \cdots * s_n$, et $(v_1, \ldots, v_n) \in \mathcal{T}_{s_1}(\mathcal{C}) \times \cdots \times \mathcal{T}_{s_n}(\mathcal{C})$ tels que $v = c(v_1, \ldots, v_n)$. De plus, par exemption de p pour v, on sait que $p \not\preccurlyeq v$ et, pour tout $i \in [1, n]$, v_i est exempt de p. D'où, $v \notin [\![p]\!]$ et, pour tout $i \in [1, n]$, $v_i \in [\![z_i s_i^{-p}]\!]$. Par conséquent, on a bien $v \in [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p}) \setminus p]\!]$.

- Le reste de la preuve est immédiat par définition du filtrage par motif et de la sémantique close.

□

On veut ainsi pouvoir établir des propriétés d'Exemption de Motif par comparaison directe entre les sémantiques des termes et motifs considérés. D'après la Proposition 3.7, l'Exemption de Motif se traduit par une caractéristique syntaxique de toutes les sous-valeurs du terme considéré. On introduit donc, dans un premier temps, une notion de sémantique fermée par relation de sous-terme :

Définition 3.12 (Sémantique profonde)**.** *Soit $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \mathcal{P}(\mathcal{C}, \mathcal{X})$, on définit la sémantique profonde de t, notée $\{\!|t|\!\}$, comme la fermeture de la sémantique close t par la relation de sous-terme : $\{\!|t|\!\} = \{v_{|\omega} \mid v \in [\![t]\!], \omega \in \mathcal{P}os(v)\}$.*

On peut maintenant écrire le corollaire de la Proposition 3.7 avec la notion de sémantique profonde, qui se traduit par une simple comparaison de sémantique :

Corollaire 3.9. *Soit un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et un motif $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$ linéaire, t est exempt de p si et seulement si $\{\!|t|\!\} \cap [\![p]\!] = \emptyset$.*

Démonstration. Soient $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et p un motif étendu. Par définition, on a $\{\!|t|\!\} = \{v_{|\omega} \mid v \in [\![t]\!], \omega \in \mathcal{P}os(v)\}$, d'où $\{\!|t|\!\} \cap [\![p]\!] = \emptyset$ si et seulement si $\forall v \in [\![t]\!], \forall \omega \in \mathcal{P}os(v), v_{|\omega} \notin [\![p]\!]$, *i.e.* $p \not\preccurlyeq v_{|\omega}$. Donc $\{\!|t|\!\} \cap [\![p]\!] = \emptyset$ si et seulement si $\forall v \in [\![t]\!]$, v est exempt de p. Par conséquent, d'après Proposition 3.7, $\{\!|t|\!\} \cap [\![p]\!] = \emptyset$ si et seulement si t est exempt de p. □

De plus, comme pour la sémantique close, on peut obtenir une décomposition récursive de la sémantique profonde d'un terme de l'algèbre :

Proposition 3.10.

- *Soit $x_s \in \mathcal{X}$ avec $s \in \mathcal{S}$, on a*

$$\{\!|x_s|\!\} \quad = \quad \bigcup_{c \in \mathcal{C}_s} \{\!|c(z_{1s_1}, \ldots, z_{ns_n})|\!\} \quad \textit{avec } \mathcal{D}om(c) = s_1 * \cdots * s_n$$

- *Soit* $c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *avec* $c \in \mathcal{C}$

$$\{\!|c(t_1, \ldots, t_n)|\!\} \quad = \quad [\![c(t_1, \ldots, t_n)]\!] \quad \cup \quad \bigcup_{i=1}^{n} \{\!|t_i|\!\}$$

- *Soit un terme* $\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *avec* $\varphi \in \mathcal{D}^n$*, on a :*

$$\{\!|\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)|\!\} \quad = \quad \{\!|x_s^{-\triangleright^{\mathcal{P}}(t_1,\ldots,t_n)}|\!\}$$

Démonstration. On prouve en Annexe A le Lemme suivant :

Lemme 3.11. *Soit* $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *avec* $c \in \mathcal{C}$*, pour tout* $i \in [1, n]$*, on a* $\forall v \in [\![t_i]\!]$*,* $\exists\, c(w_1, \ldots, w_n) \in [\![t]\!]$ *tel que* $w_i = v$*.*

On peut maintenant prouver la Proposition :

- Soit $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}$, on montre que :

$$\{\!|c(t_1, \ldots, t_n)|\!\} \quad = \quad [\![c(t_1, \ldots, t_n)]\!] \quad \cup \quad \bigcup_{i=1}^{n} \{\!|t_i|\!\}$$

  On considère $v \in \{\!|c(t_1, \ldots, t_n)|\!\}$, *i.e.* il existe $w = c(w_1, \ldots, w_n) \in [\![c(t_1, \ldots, t_n)]\!]$ et $\omega \in \mathcal{P}os(w)$ tels que $v = w_{|\omega}$. Si $\omega = \epsilon$, $v \in [\![c(t_1, \ldots, t_n)]\!]$. Sinon, on note $\omega = i.\omega'$ avec $i \in [1, n]$ et on peut remarquer que l'inclusion directe de l'égalité $[\![c(t_1, \ldots, t_n)]\!] = \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$, prouvée dans la Proposition 3.5, est également vraie dans le cas non-linéaire. D'où $w_i \in [\![t_i]\!]$, et comme $v = w_{i|\omega'}$, on a $v \in \{\!|t_i|\!\}$.
  Pour l'inclusion indirecte, on peut d'abord remarquer que pour tout $v \in [\![c(t_1, \ldots, t_n)]\!]$, on a $v \in \{\!|c(t_1, \ldots, t_n)|\!\}$. On considère maintenant $i \in [1, n]$ et $v \in \{\!|t_i|\!\}$, *i.e.* il existe $w \in [\![t_i]\!]$ et une position $\omega \in \mathcal{P}os(w)$ tels que $v = w_{|\omega}$. Donc il existe $c(w_1, \ldots, w_n) \in [\![t)]\!]$ tel que $w_i = w$, d'où $v \in \{\!|t|\!\}$.
- Le reste de la preuve est immédiat par Proposition 3.5.

□

L'équivalence donnée en Proposition 3.9 fournit une nouvelle condition nécessaire et suffisante à la propriété d'Exemption de Motif, permettant, avec cette décomposition récursive de la sémantique profonde, de mettre en place une méthode générale de vérification des propriétés d'Exemption de Motif :

Exemple 3.11. *On reprend les cas traités dans l'Exemple 3.9, en utilisant la Proposition 3.9 pour prouver les mêmes propriétés d'Exemption de Motif. On rappelle que* $p_{flat} = cons(lst(l_1), l_2)$*, et que les symboles définis sont annotés* $flatten^{!\mathcal{P}_1}$ *et* $concat^{!\mathcal{P}_2}$ *avec* $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ *et* $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$*.*

- *On a* $\{\!|cons(e, l)|\!\} = [\![cons(e, l)]\!] \cup \{\!|e_{\mathsf{Expr}}|\!\} \cup \{\!|l_{\mathsf{List}}|\!\}$ *et on remarque que* $cons(lst(nil), nil) \in [\![cons(e, l)]\!]$ *et* $cons(lst(nil), nil) \in [\![p_{flat}]\!]$*. Donc* $\{\!|cons(e, l)|\!\} \cap [\![p_{flat}]\!] \neq \emptyset$*, d'où* $cons(e, l)$ *n'est pas exempt de* p_{flat}*.*
- *De même, on a* $\{\!|cons(int(i), l)|\!\} = [\![cons(int(i), l)]\!] \cup [\![int(i)]\!] \cup \{\!|i_{\mathsf{Int}}|\!\} \cup \{\!|l_{\mathsf{List}}|\!\}$ *et on peut voir que* $cons(lst(nil), nil) \in \{\!|l_{\mathsf{List}}|\!\}$ *et et* $cons(lst(nil), nil) \in [\![p_{flat}]\!]$*. Donc* $\{\!|cons(int(i), l)|\!\} \cap [\![p_{flat}]\!] \neq \emptyset$*, d'où* $cons(int(i), l)$ *n'est pas exempt de* p_{flat}*.*
- *De même, on a* $\{\!|cons(int(i), nil)|\!\} = [\![cons(int(i), nil)]\!] \cup [\![int(i)]\!] \cup \{\!|i_{\mathsf{Int}}|\!\} \cup [\![nil]\!]$ *et on peut facilement montrer que* $\{\!|i_{\mathsf{Int}}|\!\} = [\![s(i)]\!] \cup [\![z]\!]$*. On a donc* $\{\!|cons(int(i), nil)|\!\} \cap [\![p_{flat}]\!] = \emptyset$*, d'où* $cons(int(i), nil)$ *est exempt de* p_{flat}*.*

- *Étant donné que l est exempt de $\bot$, on a $[\![flatten^{!\mathcal{P}_1}(l)]\!] = [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$. Par définition de la sémantique d'une variable annotée, on a clairement $[\![x_{\mathsf{List}}^{-p_{flat}}]\!] \cap [\![p_{flat}]\!] = \emptyset$, d'où $flatten^{!\mathcal{P}_1}(l)$ est exempt de p_{flat}.*
- *On peut en déduire que $[\![concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))]\!] = [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$. Et donc, de façon similaire, $concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))$ est exempt de p_{flat}.*
- *On a $[\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!] = [\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!] \cup [\![e_{\mathsf{Expr}}]\!] \cup [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$ et on remarque que $cons(lst(nil), nil) \in [\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!]$ et $cons(lst(nil), nil) \in [\![p_{flat}]\!]$. Par conséquent, on a $[\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!] \cap [\![p_{flat}]\!] \neq \emptyset$, d'où $cons(e, flatten^{!\mathcal{P}_1}(l))$ n'est pas exempt de p_{flat}.*
- *Enfin, étant donné que $cons(lst(l_1), l_2)$ est exempt de $\bot$, on a :*
  $[\![cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))]\!] = [\![cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))]\!] \cup [\![int(i_{\mathsf{Int}})]\!] \cup [\![s(i)]\!] \cup [\![z]\!] \cup [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$
  On a donc $[\![cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))]\!] \cap [\![p_{flat}]\!] = \emptyset$, et par conséquent, on peut conclure que $cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))$ est exempt de p_{flat}.

On proposera dans le Chapitre 4 une méthode systématique permettant de vérifier que l'intersection considérée dans la Proposition 3.9 est bien vide, reposant sur la réduction de conjonction de motifs étendus. Dans ce contexte, étant donné un terme de $\mathcal{T}(\mathcal{F}, \mathcal{X})$, on propose de se servir de la notion de variable annotée pour construire un motif ayant une sémantique close (et donc profonde) égale au terme considéré.

3.2.3 Équivalent sémantique

En partant de l'observation que la sémantique d'un terme ayant un symbole défini comme symbole de tête dépend uniquement des profils donnés en annotation du symbole et des propriétés d'Exemption de Motif vérifiées par ses sous-termes, on peut ramener l'étude de sa sémantique à l'étude de la sémantique d'un motif étendu. En effet, comme démontré dans le Proposition 3.5, cette sémantique est équivalente à la sémantique d'une variable annotée par le *motif annotant* et on peut généraliser cette approche en construisant, pour tout terme de l'algèbre, un motif étendu ayant une sémantique équivalente :

Définition 3.13 (Équivalent sémantique)**.** *Étant donné un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on appelle équivalent sémantique de t le terme $\tilde{t} \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, construit à partir de t :*

- *si $t = c(t_1, \ldots, t_n)$ avec $c \in \mathcal{C}^n$, alors $\tilde{t} = c(\tilde{t_1}, \ldots, \tilde{t_n})$;*
- *si $t = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi \in \mathcal{D}^n$, alors $\tilde{t} = z_s^{-\diamond^{\mathcal{P}}(t_1, \ldots, t_n)}$;*
- *si $t = x_s$, alors $t = x_s^{-\bot}$.*

où $z_s^{-\diamond^{\mathcal{P}}(t_1, \ldots, t_n)}$ est une variable fraiche et $\diamond^{\mathcal{P}}(t_1, \ldots, t_n) = \sum_{q \in \mathcal{Q}}$ avec $\mathcal{Q} = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ t.q. $\forall i \in [1, n], [\![\tilde{t_i}]\!] \cap [\![l_i]\!] = \emptyset\}$.

Étant donné un terme de l'algèbre et son équivalent sémantique, ils ont la même sémantique close :

Proposition 3.12. *Soit $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on a $[\![t]\!] = [\![\tilde{t}]\!]$.*

Démonstration. On prouve en Annexe A le Lemme suivant :

Lemme 3.13. *Soient un terme $t \in \mathcal{T}(\mathcal{F},\mathcal{X}) \setminus \mathcal{T}(\mathcal{C},\mathcal{X})$ et une valeur $u \in \mathcal{T}(\mathcal{C})$, $u \in [\![t]\!]$ si et seulement si il existe une position $\omega \in \mathcal{S}td\,(t) = \{\omega \in \mathcal{P}os(t) \mid \forall \omega' < \omega, t(\omega') \in \mathcal{C}\}$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\ldots,t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1,\ldots,t_n)$, telles que $u \in [\![t\,[v]_\omega]\!]$.*

On peut maintenant démontrer la Proposition par induction sur la forme de t et le nombre k de symboles définis de t dont la position est dans $\mathcal{S}td\,(t)$.

Si $t = c$ avec $c \in \mathcal{C}^0$ ou $t = x \in \mathcal{X}$, la proposition est clairement vérifiée.

Si $t = f_s^{!\mathcal{P}}(t_1,\ldots,t_n)$ avec $f \in \mathcal{D}^n$, par induction on $[\![t_i]\!] = [\![\tilde{t_i}]\!]$ pour tout $i \in [1,n]$. Avec Proposition 3.9, on a $p := \triangleright^{\mathcal{P}}(t_1,\ldots,t_n) = \diamond^{\mathcal{P}}(t_1,\ldots,t_n)$, et d'après le Lemme 3.13, $[\![t]\!] = [\![x_s^{-p}]\!] = [\![\tilde{t}]\!]$.

On considère maintenant $t = c(t_1,\ldots,t_n)$ avec $c \in \mathcal{C}^n$ et on procède par induction sur k le nombre de symboles définis de t dont la position est dans $\mathcal{S}td\,(t)$. Si $k = 0$, on a $t \in \mathcal{T}(\mathcal{C},\mathcal{X})$ et $t = \tilde{t}$, donc la Proposition est vérifiée.

On suppose maintenant $k > 0$ tel que pour tout terme u ayant strictement moins de k symboles définis à des positions de $\mathcal{S}td\,(u)$, on a $[\![u]\!] = [\![\tilde{u}]\!]$.

On considère dans un premier temps $w \in [\![t]\!]$. D'après le Lemme 3.13, il existe une position $\omega \in \mathcal{S}td\,(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\ldots,t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $p := \triangleright^{\mathcal{P}}(t_1,\ldots,t_n)$, telles que $w \in [\![t\,[v]_\omega]\!]$. On a donc, d'après l'hypothèse d'induction, $w \in [\![t\,[\tilde{v}]_\omega]\!]$, *i.e.* il existe une substitution σ telle que $w = \sigma(t\,[\tilde{v}]_\omega)$ et pour toute variable $x^{-q} \in \mathcal{V}ar(t\,[\tilde{v}]_\omega)$, $\sigma(x^{-q})$ est exempt de q. Par construction, on a $t\,[\tilde{v}]_\omega = \tilde{t}\,[v]_\omega$, et comme $p = \triangleright^{\mathcal{P}}(t_1,\ldots,t_n) = \diamond^{\mathcal{P}}(t_1,\ldots,t_n)$ par induction sur la forme t, en définissant $\sigma' = \{z_s^{-p} \mapsto v, \forall x^{-q} \in \mathcal{V}ar(t\,[\tilde{v}]_\omega).x^{-q} \mapsto \sigma(x^{-q})\}$, avec z une variable fraîche, on observe que $w \in [\![\tilde{t}\left[z_s^{-p}\right]_\omega]\!] = [\![\tilde{t}]\!]$.

On considère maintenant $w \in [\![\tilde{t}]\!]$, et une position $\omega \in \mathcal{S}td\,(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\ldots,t_n)$. Par construction de l'équivalent sémantique, $\tilde{t}_{|\omega} = x_s^{-p}$ avec $p := \triangleright^{\mathcal{P}}(t_1,\ldots,t_n) = \diamond^{\mathcal{P}}(t_1,\ldots,t_n)$ par induction sur la forme t, et x une variable fraîche. Par définition de la sémantique close et du filtrage, on a donc $x_s^{-p} \not\prec w_{|\omega}$, *i.e.* $w_{|\omega}$ est exempt de p, et $\tilde{t}\left[w_{|\omega}\right]_\omega \not\prec w$, *i.e.* $w \in [\![\tilde{t}\left[w_{|\omega}\right]_\omega]\!]$. De plus, par construction $\tilde{t}\left[w_{|\omega}\right]_\omega = t\left[\tilde{w_{|\omega}}\right]_\omega$, et d'après l'hypothèse d'induction, on a $[\![t\left[\tilde{w_{|\omega}}\right]_\omega]\!] = [\![t\left[w_{|\omega}\right]_\omega]\!]$. On a donc $w \in [\![t\left[w_{|\omega}\right]_\omega]\!] \subseteq [\![t]\!]$.

Par induction, on a donc $[\![t]\!] = [\![\tilde{t}]\!]$ pour tout $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. □

Comme la sémantique profonde est définie par fermeture de la sémantique close par relation de sous-terme, le corollaire est également vrai pour la sémantique profonde

Corollaire 3.14. *Soit $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, on a $\{\!|t|\!\} = \{\!|\tilde{t}|\!\}$.*

Démonstration. La preuve est immédiate par Proposition 3.12 et par définition de la sémantique profonde. □

L'utilisation de l'équivalent sémantique permet ainsi de simplifier les notations et surtout de ramener toute étude de sémantique à l'étude de motifs étendus :

Exemple 3.12. *On considère l'algèbre de listes introduite dans l'Exemple 3.5. On note $p_{flat} = cons(lst(l_1), l_2)$, et on rappelle que les symboles définis sont annotés $flatten^{!\mathcal{P}_1}$ et $concat^{!\mathcal{P}_2}$ avec $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ et $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$.*

- *Étant donné que $\{\!|l_{\mathsf{List}}^{-\bot}|\!\} \cap [\![\bot]\!] = \emptyset$, on a $\diamond^{\mathcal{P}_1}((l)) = p_{flat}$, d'où $[\![flatten^{!\mathcal{P}_1}(l)]\!] = [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$, i.e. la sémantique de $flatten^{!\mathcal{P}_1}(l)$ est l'ensemble des valeurs de sorte List exempte de p_{flat}.*

- *Comme* $[\![x_{\mathsf{List}}^{-p_{flat}}]\!] \cap [\![p_{flat}]\!] = \emptyset$, *on a* $\diamond^{\mathcal{P}_2}\left((flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))\right) = p_{flat}$, *et on peut en déduire que* $[\![concat^{!\mathcal{P}_2}(flatten^{!\mathcal{P}_1}(l_1), flatten^{!\mathcal{P}_1}(l_2))]\!] = [\![x_{\mathsf{List}}^{-p_{flat}}]\!]$.
- *De même, on a* $[\![cons(e, flatten^{!\mathcal{P}_1}(l))]\!] = [\![cons(e, x_{\mathsf{List}}^{-p_{flat}})]\!]$.
- *Enfin, comme* $[\![cons(lst(l_1), l_2)]\!] \cap [\![\bot]\!] = \emptyset$, $[\![cons(int(i), flatten^{!\mathcal{P}_1}(cons(lst(l_1), l_2)))]\!] = [\![cons(int(i), x_{\mathsf{List}}^{-p_{flat}}))]\!]$.

On verra dans le Chapitre 4, comment la sémantique profonde d'un motif étendu peut être décomposée en une union de sémantiques closes. Comme une intersection de sémantiques closes est équivalente à une conjonction de motifs (cf. Proposition 3.8), vérifier que l'intersection considérée dans la Proposition 3.9 est vide pourra ainsi se ramener à l'étude de la sémantique de conjonctions de motifs.

Les notions d'Exemption de Motif et de sémantique close reposent toutes deux sur une sur-approximation des formes normales potentielles des termes considérés. Dans le formalisme considéré, ces formes normales sont obtenues par une relation de réécriture définie par un CBTRS donné. On cherche donc, dans un premier temps, à observer comment ces deux notions se comportent dans le contexte de la réécriture de terme.

3.3 Application à l'analyse de Systèmes de Réécriture

Les notions d'Exemption de Motif et de sémantique close, introduites dans les sections précédentes, sont définies à partir des annotations données aux différents symboles définis de l'algèbre. Ces annotations décrivent le comportement attendu de la réduction du symbole défini considéré, en donnant un ensemble de pré-conditions et de post-conditions sur la forme des termes considérés et du résultat de la réduction. Ce dernier étant obtenu par une relation de réécriture, il est nécessaire de vérifier que le système de réécriture sous-jacent est en accord avec les annotations données.

3.3.1 Présentation de l'approche d'analyse

Étant donné un symbole défini $\varphi^{!\mathcal{P}} : s_1 * \cdots * s_n \mapsto s$, tel que le profil $l_1 * \cdots * t_n \mapsto r$ est dans l'annotation $\mathcal{P}$, ce profil sous-entend qu'étant donnés des termes clos $t_1, \ldots, t_n$, de sorte respective $s_1, \ldots, s_n$, si $t_1, \ldots, t_n$ sont respectivement exempts de $l_1, \ldots, l_n$, alors $\varphi^{!\mathcal{P}}(t_1, \ldots, t_n)$ doit être réduit en un terme exempt de r. On retrouve bien l'idée de pré-condition s'appliquant sur les arguments de la fonction, *i.e.* les termes en argument du symbole défini, pour le membre gauche du profil tandis que le membre droit donne une post-condition à satisfaire pour le résultat de la fonction. De plus, par définition de l'Exemption de Motif, si $t_1, \ldots, t_n$ sont respectivement exempts de $l_1, \ldots, l_n$, alors $\varphi^{!\mathcal{P}}(t_1, \ldots, t_n)$ est lui-même exempt de r. Donc dire qu'il doit être réduit en un terme exempt de r revient à dire que la relation de réécriture préserve les propriétés d'Exemption de Motif.

Cette approche se généralise naturellement à la sémantique close du terme : on cherche à vérifier que le CBTRS considéré induit une relation de réécriture préservant la sémantique.

Exemple 3.13. *On considère l'algèbre de termes introduite dans l'Exemple 3.1. Dans cette algèbre, on considère des termes représentant des λ-expressions pouvant contenir des constructions let, considérées comme un sucre syntaxique. On introduit donc le symbole défini* $removeLet^{!\mathcal{P}}$: $\mathsf{Expr} \mapsto \mathsf{Expr}$ *représentant la fonction transformant une telle λ-expression par une λ-expression équivalente en remplaçant toutes les constructions let par leur équivalent syntaxique. Le symbole*

removeLet est donc annoté avec $\mathcal{P} = \{\bot \mapsto let(n, e_1, e_2)\}$. *Le comportement de cette fonction est décrit par le CBTRS* $\mathcal{R}$ *suivant :*

$$\begin{cases} removeLet(var(n)) & \twoheadrightarrow & var(n) \\ removeLet(let(n, e_1, e_2)) & \twoheadrightarrow & apply(lambda(n, removeLet(e_2)), removeLet(e_1)) \\ removeLet(lambda(n, e)) & \twoheadrightarrow & lambda(n, removeLet(e)) \\ removeLet(apply(e_1, e_2)) & \twoheadrightarrow & apply(removeLet(e_1), removeLet(e_2)) \end{cases}$$

Dans ce contexte, la sémantique d'un terme de la forme $removeLet(e)$ *avec* e *un terme clos de l'algèbre est* $[\![removeLet(e)]\!] = [\![x^{-let(n,e_1,e_2)}_{\mathsf{Expr}}]\!]$, *i.e. l'ensemble des valeurs exemptes de* $let(n, e_1, e_2)$. *Elle peut être décrite comme le langage des termes de sorte* Expr *ne contenant pas le symbole let.*

Comme expliqué ci-dessus, ce terme étant exempt de $let(n, e_1, e_2)$, *on veut que cette Exemption de Motif soit préservée par réécriture, i.e. pour tout terme* u *tel que* $removeLet(e) \Longrightarrow^*_{\mathcal{R}} u$, u *doit être exempt de* $let(n, e_1, e_2)$. *Ici, on a par exemple :*

$$removeLet(let(z, lambda(s(z), var(s(z))), var(z))) \xrightarrow{\mathcal{R}} apply(lambda(z, removeLet(var(z))), removeLet(var(s(z))))$$

Et on peut en effet observer que $apply(lambda(z, removeLet(var(z))), removeLet(var(s(z))))$ *est bien exempt de* $let(n, e_1, e_2)$. *En termes de sémantique, comme d'après la Proposition 3.7, on a* u *exempt de* $let(n, e_1, e_2)$ *si et seulement si* $[\![u]\!] \subseteq [\![x^{-let(n,e_1,e_2)}_{\mathsf{Expr}}]\!] = [\![removeLet(e)]\!]$, *cela revient à prouver que la relation de réécriture induite par* $\mathcal{R}$ *préserve la sémantique.*

On veut donc introduire une méthode permettant de vérifier, pour un CBTRS donné, que la relation de réécriture induite préserve la sémantique. On propose dans la sous-section suivante, une condition suffisante à cette préservation reposant uniquement sur les profils donnés en annotation et les règles de réécriture individuelles composant le CBTRS considéré.

3.3.2 Préservation par relation de réécriture

Pour prouver la préservation de la sémantique par réécriture, on propose, dans un premier temps, d'observer l'évolution de la sémantique pour chaque règle du système considéré, dans le contexte d'une relation de réécriture. Par définition, une règle de réécriture $ls \twoheadrightarrow rs$ s'applique en transformant, pour toute substitution σ, un terme $\sigma(ls)$ en $\sigma(rs)$. On considère donc qu'une règle de réécriture préserve la sémantique si la sémantique du dernier est toujours inclue dans la sémantique du premier, quelque soit la substitution σ :

Définition 3.14 (Préservation de sémantique)**.** *Soit une règle de réécriture* $ls \twoheadrightarrow rs$, *on dit que la règle préserve la sémantique si et seulement si, pour toute substitution* σ, *on a* $[\![\sigma(rs)]\!] \subseteq [\![\sigma(ls)]\!]$.

On dit qu'un TRS $\mathcal{R}$ *préserve la sémantique si et seulement si toutes les règles de* $\mathcal{R}$ *préservent la sémantique.*

Concrètement, cela revient à vérifier que, pour toutes les applications en tête possibles d'une des règles du TRS considéré, la sémantique est préservée. En pratique, grâce aux propriétés de la sémantique close, un TRS préservant ainsi la sémantique induit une relation de réécriture qui préserve également la sémantique :

Proposition 3.15. *Soit un TRS* $\mathcal{R}$ *préservant la sémantique, on a pour tous termes clos* $t, v \in \mathcal{T}(\mathcal{F})$ *:*

$$t \Longrightarrow^*_{\mathcal{R}} v \implies [\![v]\!] \subseteq [\![t]\!]$$

Démonstration. On prouve en Annexe A le Lemme suivant :

Lemme 3.16. *Soit un terme clos $t \in \mathcal{T}(\mathcal{F})$, pour toute position $\omega \in \mathcal{P}os(t)$ et pour tout termes clos $u, v \in \mathcal{T}_s(\mathcal{F})$ avec $t_{|\omega} : s$, si $[\![u]\!] \subseteq [\![v]\!]$ alors $[\![t\,[u]_\omega]\!] \subseteq [\![t\,[v]_\omega]\!]$.*

On considère maintenant un TRS $\mathcal{R}$ préservant la sémantique, et des termes clos $t, v \in \mathcal{T}(\mathcal{F})$ tels que $t \Longrightarrow_{\mathcal{R}} v$. Par définition de la relation de réécriture, il existe une règle $ls \twoheadrightarrow rs \in \mathcal{R}$, une position $\omega \in \mathcal{P}os(t)$ et une substitution σ telles que $t_{|\omega} = \sigma(ls)$ et $v = t\,[rs]_\omega$. De plus, par préservation de la sémantique de la règle $ls \twoheadrightarrow rs$, on a $[\![\sigma(rs)]\!] \subseteq [\![\sigma(ls)]\!]$, et le résultat précédent garantit donc que $[\![v]\!] \subseteq [\![t]\!]$.

Par fermeture transitive, on obtient ainsi la propriété voulue. □

Et par extension, la relation de réécriture, induite par un système de réécriture préservant la sémantique, préserve également les propriétés d'exemption de Motif :

Corollaire 3.17. *Soient un TRS $\mathcal{R}$ préservant la sémantique, un motif étendu p et des termes clos $t, v \in \mathcal{T}(\mathcal{F})$ tels que $t \Longrightarrow^*_{\mathcal{R}} v$, on a :*

$$t \text{ est exempt de } p \implies v \text{ est exempt de } p$$

Démonstration. La preuve est immédiate par Propositions 3.15 et 3.7. □

La préservation de sémantique des règles du TRS considéré, et par extension du TRS lui-même, fournit donc bien une condition suffisante à la préservation de la sémantique, et par extension des propriétés d'Exemption de Motif, de la relation de réécriture induite.

Exemple 3.14. *Étant donné l'algèbre de termes et le CBTRS $\mathcal{R}$ considérés dans Exemple 3.13, on peut vérifier que les membres droits de chaque règle sont exempts de $p_{let} = let(n, e_1, e_2)$:*

- *On a $\{\!|var(n)|\!\} = [\![var(n)]\!] \cup [\![s(i)]\!] \cup [\![z]\!]$, on peut donc vérifier que $\{\!|var(n)|\!\} \cap [\![p_{let}]\!] = \emptyset$, et conclure que $var(n)$ est exempt de p_{let}.*
- *De même, on a :*
  $\{\!|apply(lambda(n, removeLet(e_2)), removeLet(e_1))|\!\} = \{\!|apply(lambda(n, x^{-p_{let}}_{\mathsf{Expr}}), y^{-p_{let}}_{\mathsf{Expr}})|\!\}$
  $= [\![apply(lambda(n, x^{-p_{let}}_{\mathsf{Expr}}), y^{-p_{let}}_{\mathsf{Expr}})]\!] \cup [\![lambda(n, x^{-p_{let}}_{\mathsf{Expr}})]\!] \cup [\![s(i)]\!] \cup [\![z]\!] \cup \{\!|x^{-p_{let}}_{\mathsf{Expr}}|\!\}$
  On peut donc vérifier que $\{\!|apply(lambda(n, removeLet(e_2)), removeLet(e_1))|\!\} \cap [\![p_{let}]\!] = \emptyset$, et conclure que $apply(lambda(n, removeLet(e_2)), removeLet(e_1))$ est exempt de p_{let}.
- *De même, on a :*
  $\{\!|lambda(n, removeLet(e))|\!\} = \{\!|lambda(n, x^{-p_{let}}_{\mathsf{Expr}})|\!\}$
  $= [\![lambda(n, x^{-p_{let}}_{\mathsf{Expr}})]\!] \cup [\![s(i)]\!] \cup [\![z]\!] \cup \{\!|x^{-p_{let}}_{\mathsf{Expr}}|\!\}$
  On peut donc vérifier que $\{\!|lambda(n, removeLet(e))|\!\} \cap [\![p_{let}]\!] = \emptyset$, et conclure que $lambda(n, removeLet(e))$ est exempt de p_{let}.
- *De même, on a :*
  $\{\!|apply(removeLet(e_1), removeLet(e_2))|\!\} = \{\!|apply(x^{-p_{let}}_{\mathsf{Expr}}, y^{-p_{let}}_{\mathsf{Expr}})|\!\}$
  $= [\![apply(x^{-p_{let}}_{\mathsf{Expr}}, y^{-p_{let}}_{\mathsf{Expr}})]\!] \cup \{\!|x^{-p_{let}}_{\mathsf{Expr}}|\!\}$
  On peut donc vérifier que $\{\!|apply(removeLet(e_1), removeLet(e_2))|\!\} \cap [\![p_{let}]\!] = \emptyset$, et conclure que $apply(removeLet(e_1), removeLet(e_2))$ est exempt de p_{let}.

De plus, d'après Proposition 3.5, on a pour toute règle $ls \twoheadrightarrow rs$ du système $\mathcal{R}$ et pour toute substitution σ, $[\![\sigma(ls)]\!] = [\![x^{-p_{let}}_{\mathsf{Expr}}]\!]$ et $\sigma(rs)$ est exempt de p_{let}. On peut donc conclure que toutes les règles préservent la sémantique, et d'après Proposition 3.15, la relation de réécriture préserve la sémantique. Ainsi, le système $\mathcal{R}$ étant complet et terminant, pour toute valeur $e \in \mathcal{T}_{\mathsf{Expr}}(\mathcal{C})$,

removeLet(e) *sera réduit en une valeur exempte de* p_{let}*, i.e. une valeur ne contenant pas le constructeur let.*

On peut noter que la Proposition 3.15 fournit une condition suffisante mais non nécessaire à la préservation de la sémantique. En effet, cette dernière dépend de la sur-approximation induite par les annotations données aux symboles définis, et une sur-approximation trop grossière ne permettra donc, peut-être, pas de vérifier la préservation de la sémantique. Cela dit, plus les annotations seront précises, moins la sur-approximation sera grossière, réduisant ainsi les possibilités d'un faux négatif.

Exemple 3.15. *On considère l'algèbre de termes introduite dans l'Exemple 3.5. On note* $p_{flat} = cons(lst(l_1), l_2)$*, et on considère ici que les symboles définis sont annotés* $flatten^{!\mathcal{P}_1}$ *et* $concat^{!\mathcal{P}_2}$ *avec* $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ *et* $\mathcal{P}_2 = \emptyset$*. Le comportement des fonctions représentées par ces symboles définis est décrit par le CBTRS* $\mathcal{R}$ *suivant :*

$$\begin{cases} flatten(nil) & \twoheadrightarrow & nil \\ flatten(cons(int(n), l)) & \twoheadrightarrow & cons(int(n), flatten(l)) \\ flatten(cons(lst(l), l')) & \twoheadrightarrow & concat(flatten(l), flatten(l')) \\ concat(cons(e, l), l') & \twoheadrightarrow & cons(e, concat(l, l')) \\ concat(nil, l) & \twoheadrightarrow & l \end{cases}$$

On peut vérifier qu'avec les profils choisis en annotation, la règle $flatten(cons(lst(l), l')) \twoheadrightarrow concat(flatten(l), flatten(l'))$ *ne préserve pas la sémantique. En effet, pour toute substitution* σ*,* $[\![\sigma(flatten(cons(lst(l), l')))]\!] = [\![x_{\mathsf{Expr}}^{-p_{flat}}]\!]$ *et* $[\![\sigma(concat(flatten(l), flatten(l')))]\!] = [\![x_{\mathsf{Expr}}^{-\bot}]\!]$*.*

Cela dit, on a vu dans l'Exemple 3.9, qu'en prenant $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$*, le terme* $concat(flatten(l), flatten(l'))$ *est bien exempt de* p_{flat}*. La règle préserve alors la sémantique, et on proposera dans le chapitre suivant une méthode pour vérifier que les règles de concat préservent également la sémantique.*

En pratique, on propose de considérer pour chaque règle de réécriture du CBTRS les profils annotant le symbole défini en tête du membre gauche de la règle est de vérifier que la règle respecte les propriétés d'Exemption de Motif induite par chaque profil :

Définition 3.15 (Satisfaction de profil)**.** *Soit une règle de réécriture* $\varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$*, et un profil* $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$*, on dit que la règle satisfait le profil* π *si et seulement si pour toute substitution* σ*, on a :*

$$\sigma(ls_i) \text{ est exempt de } p_i, \text{ pour tout } i \in [1, n] \implies \sigma(rs) \text{ est exempt de } p$$

La satisfaction de tous les profils annotant le symbole défini en tête du membre gauche d'une règle est en effet une condition nécessaire et suffisante à la préservation de la sémantique :

Proposition 3.18. *Soit une règle de réécriture* $\varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$*, la règle préserve la sémantique si et seulement si elle satisfait tous les profils de* $\mathcal{P}$*.*

Démonstration. Soit une règle de réécriture $ls \twoheadrightarrow rs$ avec $ls = \varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n)$. On suppose dans un premier temps que la règle satisfait tous les profils de $\mathcal{P}$, et on prouve que, pour toute substitution σ, on a $[\![\sigma(rs)]\!] \subseteq [\![\sigma(ls)]\!]$. Pour toute substitution σ, on a, d'après Proposition 3.5, $[\![\sigma(ls)]\!] = [\![\varphi_s^{!\mathcal{P}}(\sigma(ls_1), \ldots, \sigma(ls_n))]\!] = [\![x_s^{-p}]\!]$ avec $p := \triangleright^{\mathcal{P}}(\sigma(ls_1), \ldots, \sigma(ls_n))$. De plus, la règle satisfait tous les profils de $\mathcal{P}$, *i.e.* pour tout profil $l_1 * \ldots * l_n \mapsto r$ si, pour tout $i \in [1, n]$, $\sigma(l_i)$ est exempt de l_i, alors $\sigma(rs)$ est exempt de r, donc, par composition de l'Exemption de Motif

avec l'opérateur de disjonction de motif $+$, $\sigma(rs)$ est exempt de p. On peut donc conclure, avec Proposition 3.7, que $[\![\sigma(rs)]\!] \subseteq [\![x_s^{-p}]\!] = [\![\sigma(ls)]\!]$.

On suppose maintenant qu'il existe un profil $l_1 * \ldots * l_n \mapsto r \in \mathcal{P}$ qui n'est pas satisfait par la règle, *i.e.* il existe une substitution σ telle que $\sigma(ls_i)$ est exempt de l_i, pour tout $i \in [1, n]$, et $\sigma(rs)$ n'est pas exempt de r. D'après Proposition 3.5, on a donc $[\![\sigma(\varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n))]\!] \subseteq [\![x_s^{-r}]\!]$ et, avec Proposition 3.7, on sait qu'il existe $v \in [\![\sigma(rs)]\!]$ tel que v n'est pas exempt de r. D'où $[\![\sigma(rs)]\!] \not\subseteq [\![\sigma(ls)]\!]$, *i.e.* $ls \twoheadrightarrow rs$ ne préserve pas la sémantique. □

Comparer les sémantiques de deux termes est une opération compliquée alors, qu'avec la Proposition 3.9, on a une condition nécessaire et suffisante facilement vérifiable, avec l'aide des décompositions de sémantique explicitées dans Propositions 3.5 et 3.10 pour établir des propriétés d'Exemption de Motif. On proposera dans le chapitre suivant une méthode systématique, utilisant l'ensemble des notions introduites ici, pour vérifier des propriétés d'Exemption de Motif. Enfin, on peut remarquer que les propriétés voulues, dans le contexte de la Définition 3.15, dépendent de substitutions vérifiant les pré-conditions induites par les profils considérés. On proposera donc également une méthode permettant d'inférer la forme des termes obtenus par une telle substitution.

3.4 Discussion sur l'Exemption pour des Systèmes non-linéaires

Dans le cas de CBTRS non-linéaires, les membres droits des règles ne sont pas nécessairement linéaires. On a en effet vu dans les Sections 3.1 et 3.2, que les notions d'Exemption de Motif et de sémantique n'étaient préservées par substitution que pour des termes linéaires. Ainsi, dans le contexte d'un CBTRS, il serait difficile de conclure quant aux notions de préservation de sémantique (Définition 3.14) et de satisfaction de profil (Définition 3.15) qui doivent être vérifiées pour toute substitution.

La principale raison pour laquelle les notions d'Exemption de Motif et de sémantique sont ainsi limitées dans le contexte de termes non-linéaires est que leur définition ne fait aucune corrélation, dans un terme t considéré, entre deux sous-termes indentiques de la forme $\varphi(t_1, \ldots, t_n)$ avec $\varphi \in \mathcal{D}^n$. Comme on l'a expliqué précédemment, ces notions tentent décrire syntaxiquement la potentielle forme normale de t en donnant une sur-approximation de cette dernière basée sur les annotations des différents symboles définis contenus dans t. Cela dit chaque symbole défini est considéré indépendamment et la sur-approximation ainsi obtenue ignore le fait que, dans le cadre d'une application déterministe de la relation de réécriture induite par le CBTRS considéré (ou dans le cas où le CBTRS, lui-même, serait confluent), les deux sous-termes $\varphi(t_1, \ldots, t_n)$ identiques seront réduits de manière identique.

Exemple 3.16. *On reprend l'algèbre de termes considérée dans l'Exemple 3.7 et on étudie le terme $t = c(g(c(a, b)), g(c(a, b)))$. Étant que $g^{!\mathcal{P}_g}$ est annoté avec $\mathcal{P}_g = \emptyset$, t n'est pas exempt de $c(a, b)$ puisque $c(a, b) \in [\![t]\!]$. On remarque, en effet, que l'équivalent sémantique de t est $\tilde{t} = c(y_{\mathsf{S2}}^{-\perp}, z_{\mathsf{S2}}^{-\perp})$ puisque la sémantique, comme la propriété d'Exemption de Motif, ne considère aucune corrélation entre les deux instances de $g(c(a, b))$.*

On peut cependant argumenter qu'étant donnée une relation de réécriture confluente, ou appliquée avec une stratégie déterministe, ces deux instances seront nécessairement réduites de façon identique, i.e. la première ne peut pas être réduite à a et la deuxième à b. On peut donc raisonnablement considérer que toute potentielle forme normale de t est exempt de $c(a, b)$.

On ainsi vu avec les Propriétés 3.3 et 3.6 ne s'appliquent pas pour les termes non-linéaires, i.e. les propriétés d'Exemption de Motif et la sémantique close ne sont pas nécessairement préservées

par substitution. On observe en effet qu'avec $\sigma = \{x \mapsto g(c(a,b))\}$*, on a* $\sigma(c(x,x)) = t$ *qui n'est pas exempt de* $c(a,b)$*, bien que* $c(x,x)$ *le soit. De même, on a* $c(a,b) \in [\![t]\!]$ *mais* $c(a,b) \notin [\![c(x,x)]\!]$*.*

Afin de prouver la préservation de la sémantique (et, par extension, des propriétés d'Exemption de Motif) par réécriture, on a proposé dans la Section précédente de considérer les notions de règles préservant la sémantique (Définition 3.14) et/ou satisfaisant un profil (Définition 3.15). Ces notions considérant les propriétés syntaxiques des termes obtenus par substitution à partir des membres gauches et droits des règles considérés, il sera plus difficile de s'en servir pour analyser un CBTRS non-linéaire. On peut cependant, en première approximation, se limiter à des substitutions valeurs :

Proposition 3.19. *Étant donné un motif* $p \in \mathcal{P}(\mathcal{C},\mathcal{X})$ *linéaire et un terme* $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$*, on a, pour toute substitution valeur* ς *:*

- $[\![\varsigma(t)]\!] \subseteq [\![t]\!]$ *;*
- *si* t *est exempt de* p *alors* $\varsigma(t)$ *est exempt de* p*.*

Démonstration. Soient $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ et une substitution valeur ς.

On considère dans un premier temps $t \in \mathcal{T}(\mathcal{C},\mathcal{X})$, donc $\varsigma(t) \in \mathcal{T}(\mathcal{C},\mathcal{X})$, et $v \in [\![\varsigma(t)]\!]$. Par définition de la sémantique close, il existe une substitution σ telle $v = \sigma(\varsigma(t))$, et donc, par composition des substitutions, $v = \sigma \circ \varsigma(t)$, d'où $v \in [\![t]\!]$.

On considère maintenant $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, et on prouve par induction sur k le nombre de symboles définis dans t que $[\![\varsigma(t)]\!] \subseteq [\![t]\!]$. Si $k = 0$, $t \in \mathcal{T}(\mathcal{C},\mathcal{X})$ et on vient de montrer l'inclusion. On suppose maintenant $k > 0$ tel que pour tout terme u ayant moins de k symbole défini $[\![\varsigma(u)]\!] \subseteq [\![u]\!]$. Soit $v \in [\![\varsigma(t)]\!]$, par définition de la sémantique close, il existe une position $\omega \in \mathcal{P}os(\varsigma(t))$ telle que $\varsigma(t)_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1,\ldots,u_n)$ avec $\varphi \in \mathcal{D}^n$, et une valeur $w \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(u_1,\ldots,u_n)$ telle que $v \in [\![\varsigma(t)\,[w]_\omega]\!]$. Comme $w \in \mathcal{T}(\mathcal{C})$, on a $\varsigma(t)\,[w]_\omega = \varsigma(t\,[w]_\omega)$, donc par induction, on a $v \in [\![t\,[w]_\omega]\!]$. De plus, comme ς est une substitution valeur, on a $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\ldots,t_n)$ avec $u_i = \varsigma(t_i), \forall i \in [1,n]$. Et, pour tout profil $l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$, comme, par induction, on a $[\![\varsigma(t_i)]\!] \subseteq [\![t_i]\!], \forall i \in [1,n]$, Proposition 3.7 garantit que si t_i est exempt de l_i, alors $\varsigma(t_i)$ l'est aussi, d'où $\{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ *t.q.* $\forall i \in [1,n], t_i$ est exempt de $l_i\} \subseteq \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ *t.q.* $\forall i \in [1,n], u_i$ est exempt de $l_i\}$. Donc, par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, w est exempt de $\triangleright^{\mathcal{P}}(t_1,\ldots,t_n)$ et, par définition de la sémantique close, $[\![t\,[w]_\omega]\!] \subseteq [\![t]\!]$, d'où $v \in [\![t]\!]$.

La préservation des propriétés d'Exemption de Motif découle directement de la préservation de la sémantique, via la Proposition 3.7. □

On verra dans le Chapitre 5 qu'on peut ainsi reformuler les notions de préservation de sémantique (Définition 3.14) et de satisfaction de profil (Définition 3.15) dans le cadre d'une relation de réécriture basée sur les substitutions valeurs (*i.e.* une relation de réécriture avec une stratégie stricte). Cela revient cependant à considérer une stratégie d'évaluation en *Appel par valeur*, ce qui n'est pas forcément la stratégie universellement utilisée par tous les langages de programmation.

De plus travailler avec des termes non-linéaires peut poser d'autres difficultés, comme notamment l'évaluation de leur sémantique puisque la décomposition proposée dans les Propositions 3.5 et 3.8 n'est pas forcément applicable sur des termes non-linéaires.

On proposera dans le Chapitre 5, des définitions alternatives aux Définitions 3.3 et 3.4 permettant d'éviter de se restreindre à la considération d'une stratégie d'évaluation en *Appel par valeur*. Et on verra comment la méthode d'analyse statique de CBTRS proposée dans le Chapitre suivant peut être adaptée pour prendre en compte les contraintes liées à la non-linéarité des termes et à ces définitions alternatives.

3.5 Synthèse

Dans ce Chapitre, nous avons présenté un formalisme permettant de vérifier des propriétés de correction syntaxique des transformations, décrites par la notion d'Exemption de Motif. Formellement, cette notion exprime une propriété structurelle des termes de l'algèbre considérée, en garantissant qu'ils ne contiennent aucun sous-terme filtré par un certain motif. Elle permet ainsi de définir le système d'annotation utilisé pour décorer les symboles définis de l'algèbre : un symbole $\varphi : s_1 * \cdots * s_n \mapsto s$ peut être annoté par un ou plusieurs profil(s) de la forme $p_1 * \cdots * p_n \mapsto p$ avec $p_1, \ldots, p_n, p$ des motifs étendus, pour indiquer que toute réduction d'un terme ayant φ comme symbole de tête, avec comme argument des termes respectivement exempts de $p_1, \ldots, p_n$ doit mener à un terme exempt de p.

Ces profils décrivent ainsi des spécifications algébriques du comportement des fonctions associées. On propose alors de vérifier que le système de réécriture encodant les fonctions étudiées est conforme à ces spécifications. Pour cela, on utilise les annotations des symboles définis pour définir une généralisation de la notion de sémantique close, introduite dans [CM19], permettant d'exprimer, par une structure finie, une sur-approximation des formes normales potentielles d'un terme. Il est alors nécessaire de prouver que cette sur-approximation est bien cohérente avec le CBTRS encodant la transformation : il ne faut pas que le système permette d'obtenir un terme en dehors de cette sur-approximation, ce qui signifierait qu'il existe une réduction contredisant les profils choisis.

En pratique, cela revient à vérifier que la relation de réécriture induite par le CBTRS considéré préserve la sémantique. Pour prouver cette préservation de sémantique, on a montré (Proposition 3.18) qu'il suffisait de vérifier que chaque règle du système satisfait tous les profils décorant le symbole de tête de son membre gauche, *i.e.* pour toute substitution telle que les instances en paramètre du membre gauche de la règle satisfont la pré-condition du profil, l'instance correspondante du membre droit vérifie la post-condition. On propose d'étudier dans les Chapitres suivant une méthode d'analyse statique permettant de vérifier cette propriété, d'abord dans le cas de systèmes linéaires, puis on s'intéressera plus particulièrement aux cas non-linéaires.

4

Analyse statique par Exemption de Motif

Comme présenté dans la Section 3.1.2, choisir un profil pour annoter un symbole défini revient à faire une supposition sur les formes normales potentiellement obtenues par réécriture suivant un CBTRS considéré. Ce dernier décrit concrètement le comportement de la fonction associée au symbole annoté, et il est donc nécessaire de vérifier que ce comportement est bien en accord avec le(s) profil(s) choisi(s). On proposera donc dans ce chapitre une méthode d'analyse statique du CBTRS considéré qui permet de vérifier son adéquation avec les profils choisis.

Dans le chapitre précédent, on a vu que le comportement attendu peut être décrit comme une préservation de la sémantique par la relation de réécriture induite par le CBTRS. De plus, on a montré qu'en utilisant la notion de satisfaction de profil (Définition 3.15), le problème peut être reformulé sous la forme de propriétés d'Exemption de Motifs sur les règles du CBTRS. Ces propriétés sont, cependant, souvent difficiles à vérifier directement depuis la définition de l'Exemption de Motif (Définition 3.3), puisqu'il serait potentiellement nécessaire de vérifier un nombre infini de valeurs.

Dans le cadre d'une analyse statique, on veut proposer une méthode applicable de façon systématique, qui ne nécessite donc pas la vérification explicite d'une infinité de valeurs. On utilisera, pour cela, les sémantiques de termes, introduites dans le Chapitre précédent et leur relation avec l'Exemption de Motif (Proposition 3.9), pour établir les propriétés d'Exemption de Motif recherchées. La vérification de la satisfaction d'un profil se présente ainsi en trois étapes majeures :

1. une étape d'inférence des substitutions considérées par la notion de satisfaction de profil ;
2. une étape de construction de l'équivalent sémantique (Définition 3.13) pour ramener les termes considérés à des motifs constructeurs ;
3. une étape de calcul des sémantiques pour prouver que l'intersection de sémantiques considérée par la Proposition 3.9 est bien vide.

Pour ce faire, on commencera par présenter, dans la Section suivante, une technique de décomposition de la sémantique profonde en une union de sémantiques closes. La Section 4.2 présentera ensuite un ensemble de règles de réduction des motifs étendus, permettant notamment de vérifier qu'une intersection de sémantiques closes est vide. Ces approches définissent concrètement l'étape de calcul des sémantiques, mais sont également essentielles aux étapes d'inférence et de construction de l'équivalent sémantique qui seront détaillées dans les Sections 4.3 et 4.4. On conclura ce Chapitre avec une présentation de la méthode d'analyse ainsi définie et son application sur un cas d'étude.

On se restreint dans ce Chapitre à l'étude de systèmes de réécriture linéaires, et on étudiera plus en détails les systèmes non-linéaires dans le Chapitre suivant.

4.1 Calcul de Sémantique profonde

Vérifier des propriétés d'Exemption de Motif est un problème non-trivial : on a vu dans le Chapitre précédent que vérifier une telle propriété depuis sa Définition (3.3) nécessite en effet de considérer un nombre potentiellement infini de valeurs, et il n'est donc pas possible d'en déduire directement une méthode d'analyse statique. Pour surconvenir à ce problème, on a étendu la notion de sémantique qui avait notamment était utilisée dans [CM19] pour représenter l'ensemble, potentiellement infini, des instances d'un motif par une structure finie, munie d'opérateurs de disjonction et de différence. Cela dit, la simple forme d'un motif ne permet pas encore de conclure quant aux propriétés d'Exemption de Motif :

Exemple 4.1. *Observons le terme* $t = cons(B, cons(e, cons(A, nil)))$ *de l'algèbre de termes défini par la signature* Σ_{sorted} *présentée dans l'Exemple 3.3. Ce terme n'est clairement pas exempt de* $p_{sorted} = cons(B, cons(A, l))$ *puisque* $cons(B, cons(A, cons(A, nil))) \in [\![t]\!]$ *est filtrée par* p_{sorted}*. Mais on peut même aller plus loin, en représentant le terme* t *et le motif* p_{sorted} *comme des arbres côte à côte :*

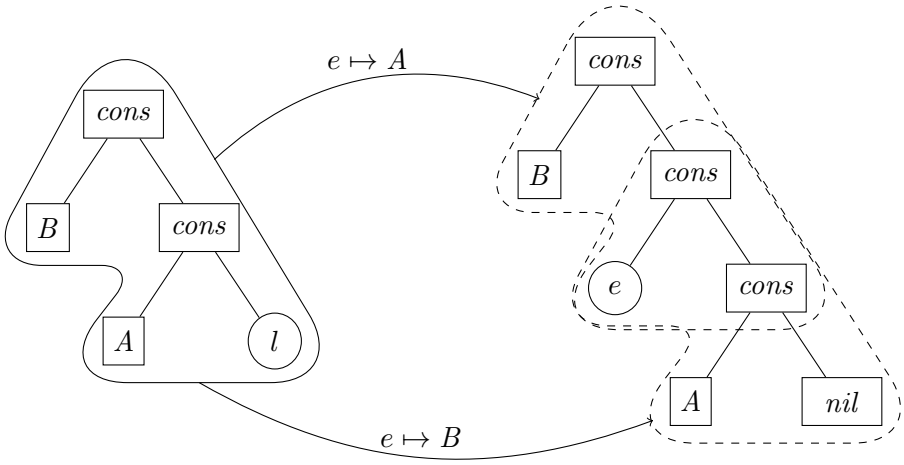


FIGURE 4.1 – Exemption de Motif et filtrage profond

On voit avec cette représentation que non seulement t *n'est pas exempt de* p_{flat} *mais également qu'aucune valeur de sa sémantique ne l'est : la variable* e *peut être instanciée soit par* A *soit par* B*, et dans le premier cas la valeur obtenue est filtrée par* p_{flat} *alors que dans le deuxième cas, le sous-terme de la branche droite est filtrée par* p_{flat}*. Dans le deuxième cas, la sémantique close ne permet cependant pas de facilement mettre en évidence la forme de ce sous-terme.*

La sémantique profonde d'un terme, définie comme la fermeture par relation de sous-terme de sa sémantique close, permet, elle, d'apporter une description beaucoup plus exhaustive de la forme d'un terme et de ses sous-termes. Cette exhaustivité permet notamment d'établir une équivalence relativement simple l'Exemption de Motif et la notion de sémantique profonde (Proposition 3.9). Mais, contrairement à la sémantique close qui fournit une équivalence naturelle

entre les opérations sur les motifs et les opérations ensemblistes (Proposition 3.8), la composition de la sémantique profonde avec les différents opérateurs considérés est plus compliquée. En particulier, les opérations de conjonction et de différence de motifs sont difficiles à exprimer au niveau de la sémantique profonde.

On propose ainsi dans cette section d'étudier la sémantique profonde de motifs étendus, et plus particulièrement de donner une décomposition de la sémantique profonde d'un motif quasi-additif (Définition 3.7) en une union de sémantiques closes.

4.1.1 Sémantique profonde de Motifs étendus

En général, étudier la sémantique profonde d'un motif peut se ramener, de façon assez naturelle, à l'étude de sa sémantique close. En effet, étant donné deux motifs ayant la même sémantique close, ils auront également, par définition, la même sémantique profonde. Et bien que [CM19] propose déjà un ensemble de règles permettant de comparer et d'étudier des sémantiques closes, les méthodes originellement présentées ne sont pas suffisantes dans le formalisme considéré ici : la sémantique close présentée dans [CM19] a été étendu dans notre formalisme pour prendre en compte les symboles définis et leurs annotations.

Pour simplifier les notations et les termes considérés, on a déjà présenté dans la Section 3.2.3, la notion d'équivalent sémantique, qui permet, pour tout terme de $\mathcal{T}(\mathcal{F}, \mathcal{X})$, de construire un terme de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ ayant la même sémantique. On verra dans la Section 4.4, comment on peut en pratique construire cet équivalent sémantique, et on se restreint donc à l'étude de la sémantique des termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$. Dans un premier temps, on peut observer, comme montré dans la Proposition 3.5, qu'un terme de la sémantique de $[\![x_s]\!]$ est un terme de la forme $c(v_1, \ldots, v_n)$ avec $c \in \mathcal{C}_s$, et donc que l'on a $[\![x_s]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(z_{1s_1}, \ldots, z_{ns_n})]\!]$. Cependant, dans le cadre de termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, il faut également prendre en compte l'annotation des variables, ce qui donne, de façon similaire $[\![x_s^{-p}]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(z_1 s_1^{-p}, \ldots, z_n s_n^{-p}) \setminus p]\!]$ (Proposition 3.8). L'introduction d'un opérateur de différence $\setminus$ supplémentaire est cependant problématique ici, puisqu'elle rendrait une méthode de réduction similaire à celle de [CM19] non terminante.

Une telle approche n'étant donc pas applicable à l'étude d'une sémantique profonde, on propose de définir une technique permettant la décomposition de la sémantique profonde d'un terme de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ comme une union de sémantiques closes. Pour cela, on généralise les termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ par des motifs quasi-symboliques, et comme on ne considère que des motifs linéaires, on observe que le nom des variables n'a pas d'importance : les variables x_s^{-p} et y_s^{-p} ont la même sémantique. On considèrera ainsi, dans la suite de ce Chapitre, que deux variables de même sorte ayant la même annotation sont donc strictement équivalentes.

On commence donc par généraliser aux motifs quasi-symboliques la décomposition récursive de la sémantique profonde, proposée dans la Proposition 3.10 pour les termes de $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$. Contrairement à la sémantique d'un terme, la sémantique d'un motif peut être vide. La décomposition de la sémantique profonde d'un motif faisant apparaître celles de ses sous-motifs, elle n'est valable que quand sa sémantique est non-vide :

Proposition 4.1. *Soit un motif quasi-symbolique linéaire de la forme $c(t_1, \ldots, t_n)$ avec $c \in \mathcal{C}^n$, on a :*

- *Si $[\![t_i]\!] \neq \emptyset, \forall i \in [1, n]$, alors $\{\!\{c(t_1, \ldots, t_n)\}\!\} = [\![c(t_1, \ldots, t_n)]\!] \cup \left(\bigcup_{i=1}^{n} \{\!\{t_i\}\!\} \right)$;*
- *sinon, $\{\!\{c(t_1, \ldots, t_n)\}\!\} = \emptyset$.*

Démonstration. Soit un motif quasi-additif linéaire de la forme $c(t_1, \ldots, t_n)$ avec $c \in \mathcal{C}^n$.

S'il existe $i \in [1,n]$ tel que $[\![t_i]\!] = \emptyset$, d'après Proposition 3.8, on a $[\![c(t_1,\ldots,t_n)]\!] = \emptyset$, d'où $\{\!\{c(t_1,\ldots,t_n)\}\!\} = \emptyset$.

Sinon, on a $[\![t_i]\!] \neq \emptyset, \forall i \in [1,n]$ et on prouve les deux inclusions séparément :

- Soit $u \in \{\!\{c(t_1,\ldots,t_n)\}\!\}$, par définition il existe $v \in [\![c(t_1,\ldots,t_n)]\!]$ et $\omega \in \mathcal{P}os(v)$ tels que $u = v_{|\omega}$. Si $\omega = \epsilon$, alors $u = v$, d'où $u \in [\![c(t_1,\ldots,t_n)]\!]$. Sinon, on peut remarquer que comme $v \in [\![c(t_1,\ldots,t_n)]\!]$, d'après Proposition 3.8, $v = c(v_1,\ldots,v_n)$ avec $v_i \in [\![t_i]\!], i \in [1,n]$. Il existe $j \in [1,n]$ tel que $\omega = j.\omega'$, donc $u = v_{i|\omega'}$, d'où $u \in \{\!\{t_j\}\!\}$. On a donc bien $u \in [\![c(t_1,\ldots,t_n)]\!] \cup (\bigcup_{i=1}^{n} \{\!\{t_i\}\!\})$.
- On peut remarquer que par définition, on a clairement $[\![c(t_1,\ldots,t_n)]\!] \subseteq \{\!\{c(t_1,\ldots,t_n)\}\!\}$. Soit $j \in [1,n]$, on considère maintenant $u \in \{\!\{t_j\}\!\}$. Par définition, il existe $v_j \in [\![t_j]\!]$ et $\omega \in \mathcal{P}os(v)$ tels que $u = v_{j|\omega}$. En prenant $v_i \in [\![t_i]\!]$, pour tout $i \neq j$, on peut construire un terme $v = c(v_1,\ldots,v_n)$ tel que $v \in [\![c(t_1,\ldots,t_n)]\!]$, d'après Proposition 3.8, et $u = v_{|j.\omega}$. On a donc bien $v \in \{\!\{c(t_1,\ldots,t_n)\}\!\}$.

□

Comme on cherche à donner une décomposition de la sémantique profonde des motifs quasi-symboliques, il reste maintenant à décomposer la sémantique profonde des motifs de la forme $x_s^{-p} \setminus r$, avec r un motif additif régulier (on peut remarquer que pour x_s^{-p}, on peut se ramener à cette forme puisque $[\![x_s^{-p}]\!] = [\![x_s^{-p} \setminus \bot]\!]$). Dans un premier temps, on observe donc qu'avec la Proposition 3.8, on a :

$$\begin{array}{rcll} [\![x_s^{-p} \setminus r]\!] & = & [\![x_s^{-p}]\!] \setminus [\![r]\!] & \\ & = & \bigcup\limits_{c \in \mathcal{C}_s} [\![c(z_{0s_1}^{-p},\ldots,z_{ns_n}^{-p}) \setminus p]\!] \setminus [\![r]\!] & \text{avec } \mathcal{D}om(c) = s_0 * \cdots * s_n \\ & = & \bigcup\limits_{c \in \mathcal{C}_s} [\![c(z_{0s_1}^{-p},\ldots,z_{ns_n}^{-p})]\!] \setminus ([\![p]\!] \cup [\![r]\!]) & \text{avec } \mathcal{D}om(c) = s_0 * \cdots * s_n \\ & = & \bigcup\limits_{c \in \mathcal{C}_s} [\![c(z_{0s_1}^{-p},\ldots,z_{ns_n}^{-p}) \setminus (p+r)]\!] & \text{avec } \mathcal{D}om(c) = s_0 * \cdots * s_n \end{array} \tag{4.1}$$

Il faut maintenant décomposer la sémantique des motifs de la forme $c(z_{1s_1}^{-p},\ldots,z_{ns_n}^{-p}) \setminus q$ où q est un motif additif et régulier. On montre donc que

Lemme 4.2. *Étant donnés une sorte s, des symboles constructeurs $c, c' \in \mathcal{C}_s$, d'arités respectives n et m, $p_1,\ldots,p_n$ et $q_1,\ldots,q_m$ des motifs, on a :*

- $[\![c(p_1,\ldots,p_n) \setminus c'(q_1,\ldots,q_m)]\!] = \begin{cases} \bigcup\limits_{i=1}^{n} [\![c(p_1,\ldots,p_i \setminus q_i,\ldots,p_n)]\!] & \text{si } c = c' \\ [\![c(p_1,\ldots,p_n)]\!] & \text{sinon} \end{cases}$;
- $[\![c(p_1,\ldots,p_n) \setminus x_s^{-\bot}]\!] = \emptyset$

Démonstration. Soient une sorte s, des symboles constructeurs $c, c' \in \mathcal{C}_s$, d'arités respectives n et m, $p_1,\ldots,p_n$ et $q_1,\ldots,q_m$ des motifs.

On considère d'abord $c = c'$, et on prouve que $[\![c(p_1,\ldots,p_n) \setminus c(q_1,\ldots,q_n)]\!] = \bigcup\limits_{i=1}^{n} [\![c(p_1,\ldots,p_i \setminus q_i,\ldots,p_n)]\!]$. On prouve les deux inclusions séparément :

- Soit $v \in [\![c(p_1,\ldots,p_n) \setminus c(q_1,\ldots,q_n)]\!]$. D'après Proposition 3.8, $v \in [\![c(p_1,\ldots,p_n)]\!]$ et $v \notin [\![c(q_1,\ldots,q_n)]\!]$, *i.e.* il existe $(v_1,\ldots,v_n) \in [\![p_1]\!] \times \cdots \times [\![p_n]\!]$ tel que $v = c(v_1,\ldots,v_n)$ et $v \not\leftarrow c(q_1,\ldots,q_n)$. Donc, il existe $j \in [1,n]$ tel que $q_j \not\leftarrow v_j$, *i.e.* $v_j \in [\![p_j \setminus q_j]\!]$. D'où $v \in [\![c(p_1,\ldots,p_j \setminus q_j,\ldots,p_n)]\!]$.

- Soit $j \in [1,n]$ et $v \in [\![c(p_1,\ldots,p_j\backslash q_j,\ldots,p_n)]\!]$. D'après Proposition 3.8, il existe $(v_1,\ldots,v_n) \in [\![p_1]\!]\times\cdots\times[\![p_j\setminus q_j]\!]\times\cdots\times[\![p_n]\!]$. Donc pour tout $i \in [1,n]$, $p_i \not\ll v_i$, d'où $c(p_1,\ldots,p_n) \not\ll v$, *i.e.* $v \in [\![c(p_1,\ldots,p_n)]\!]$. De plus, comme $v_j \in [\![p_j \setminus q_j]\!]$, $q_j \not\ll v_j$, d'où $c(q_1,\ldots,q_n) \not\ll v$, *i.e.* $v \notin [\![c(q_1,\ldots,q_n)]\!]$. On a donc $v \in [\![c(p_1,\ldots,p_n)]\!] \setminus [\![q_1,\ldots,q_n]\!] = [\![c(p_1,\ldots,p_n) \setminus [\![q_1,\ldots,q_n]]\!]$.

La preuve des deux égalités suivantes est directe grâce à la Proposition 3.8, en observant que pour tout $v \in [\![c(p_1,\ldots,p_n)]\!]$ on a $v \notin [\![c'(q_1,\ldots,q_m)]\!]$ et $v \in [\![x_s^{-\perp}]\!]$. □

De plus, on observe que, d'après Proposition 3.8, on a $[\![p \setminus (r+q)]\!] = [\![p]\!] \setminus ([\![r]\!] \cup [\![q]\!]) = ([\![p]\!]\setminus[\![r]\!])\setminus[\![q]\!] = [\![(p\backslash r)\backslash q]\!]$ et $[\![(p+q)\backslash r]\!] = ([\![p]\!]\cup[\![q]\!])\setminus[\![r]\!] = ([\![p]\!]\setminus[\![r]\!])\cup([\![q]\!]\setminus[\![r]\!]) = [\![(p\backslash r)+(q\backslash r)]\!]$. Le motif r dans l'Équation 4.1 étant additif régulier, on peut donc utiliser le Lemme 4.2 pour distribuer récursivement les sous-motifs composant r sur les différents membres de la somme.

Étant donné un constructeur $c \in \mathcal{C}^n$ et un motif additif r, on peut ainsi construire un ensemble $Q_c(r)$ de n-uplets $(\!|q_1,\ldots,q_n|\!)$, où chaque q_i est soit $\perp$ soit une somme de sous-termes de r, tel que [1] :

$$[\![c(p_1,\ldots,p_n) \setminus r]\!] \quad = \quad \bigcup_{q\in Q_c(r)} [\![c(p_1 \setminus q_1,\ldots,p_n \setminus q_n)]\!] \tag{4.2}$$

Exemple 4.2. *On considère l'algèbre de termes définie par la signature Σ_{list}, et on utilise le Lemme 4.2 pour donner une décomposition de la sémantique close de $cons(e,l)\backslash(cons(lst(l_1),l_2)+cons(e,nil)+nil)$:*

$$\begin{aligned}
&[\![cons(e,l) \setminus (cons(lst(l_1),l_2) + cons(e,nil) + nil)]\!] \\
&\quad = [\![(cons(e,l) \setminus cons(lst(l_1),l_2)) \setminus (cons(e,nil)+nil)]\!] \\
&\quad = [\![(cons(e \setminus lst(l_1),l) + cons(e,l\setminus l_2)) \setminus (cons(e,nil)+nil)]\!] \\
&\quad = [\![cons(e\setminus lst(l_1),l) \setminus cons(e,nil) \setminus nil + cons(e,l\setminus l_2) \setminus cons(e,nil) \setminus nil]\!] \\
&\quad = [\![(cons(e\setminus(lst(l_1)+e),l) + cons(e\setminus lst(l_1), l\setminus nil)) \setminus nil \\
&\qquad\quad +(cons(e\setminus e, l\setminus l_2) + cons(e,l\setminus(l_2+nil))) \setminus nil]\!] \\
&\quad = [\![cons(e\setminus(lst(l_1)+e), l\setminus\perp) + cons(e\setminus lst(l_1), l\setminus nil) \\
&\qquad\quad +cons(e\setminus e, l\setminus l_2) + cons(e\setminus\perp, l\setminus(l_2+nil))]\!]
\end{aligned}$$

On a donc, dans ce cas $Q_{cons}(cons(lst(l_1),l_2)+cons(e,nil)+nil) = \{(lst(l_1)+e,\perp),(lst(l_1),nil),(e,l_2),(\perp,l_2+nil)\}$.

Le motif r étant additif, on peut facilement le décomposer en sous motifs additifs $r_1,\ldots,r_m$, de telle façon à avoir $r = \sum_{i=1}^{m} r_i$ où chaque motif r_i est soit une variable, soit un motif ayant un constructeur comme symbole de tête. La construction de $Q_c(r)$ se faisant par simple distribution des sous-termes des motifs r_i ayant comme symbole de tête c, on peut donc automatiser la construction de $Q_c(r)$ en utilisant l'algorithme suivant :

1. On rappelle que pour un n-uplet q, on note $q = (\!|q_1,\ldots,q_n|\!)$.

Fonction `computeQc`(c, r)
  Entrées : c : symbole constructeur,
    r : motif additif
  Résultat : ensemble de tuples $Q_c(r)$
  $Q_c \longleftarrow \{(\!|\overbrace{\bot, \ldots, \bot}^{m}|\!)\}$ avec $m = arity(c)$
  pour $i = 1$ **à** n *avec* $r = \sum_{i=1}^{n} r_i$ **faire**
    si $r_i \in \mathcal{X}$ **alors**
      retourner $\emptyset$
    sinon si $r_i(\epsilon) = c$ **alors**
      $tQ_c \longleftarrow \emptyset$
      pour $(\!|q_1, \ldots, q_m|\!) \in Q_c, k \in [1, m]$ **faire**
        $tQc \longleftarrow \{(\!|q_1, \ldots, q_k + r_{i|k}, \ldots, q_m|\!)\} \cup tQc$
      $Q_c \longleftarrow tQ_c$
  retourner Q_c

FIGURE 4.2 – Calcul de l'ensemble de tuples $Q_c(r)$ pour un symbole constructeur c et un motif additif r donnés.

Pour revenir à la sémantique profonde, on a donc, avec la définition de la sémantique profonde (Définition 3.12), la Proposition 4.1 et l'Équation 4.2 :

$$
\begin{aligned}
\{\!|x_s^{-p} \setminus r|\!\} &\overset{Def}{=} \{u_{|\omega} \mid u \in [\![x_s^{-p} \setminus r]\!], \omega \in \mathcal{P}os(u)\} \\
&\overset{Eq\,4.2}{=} \left\{u_{|\omega} \mid u \in \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c(r+p)} [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!], \omega \in \mathcal{P}os(u)\right\} \\
&= \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c(r+p)} \left\{u_{|\omega} \mid u \in [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!], \omega \in \mathcal{P}os(u)\right\} \\
&\overset{Def}{=} \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c(r+p)} \{\!|c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)|\!\} \\
&\overset{Pr\,4.1}{=} \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{-p}(r+p)} [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{-p}(r+p)} \bigcup_{i=1}^{n} \{\!|x_{s_i}^{-p} \setminus q_i|\!\} \\
&= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{-p}(r+p)} \bigcup_{i=1}^{n} \{\!|x_{s_i}^{-p} \setminus q_i|\!\}
\end{aligned}
\tag{4.3}
$$

avec $Q_c^{-p}(r) = \{q = (\!|q_1, \ldots, q_n|\!) \in Q_c(p) \mid \forall i \in [1, n], [\![x_{s_i}^{-p} \setminus q_i]\!] \neq \emptyset\}$.

Étant donné qu'on a un nombre fini de sortes et que les tuples de Q_c, donnés par l'algorithme 4.2, sont des sommes de sous-termes de r et p (dont il existe aussi seulement un nombre fini), il est possible de calculer le point fixe obtenu via le développement précédent. Certains termes de sémantique profonde peuvent persister dans ce point fixe, mais on peut remarquer, via un raisonnement inductif analysant la forme des sous-termes, que ces derniers sont bien caractérisés par les termes de sémantique close. Cette approche permet ainsi de proposer une décomposition de la sémantique profonde d'un motif de la forme $x_s^{-p} \setminus r$.

Exemple 4.3. *On considère l'algèbre de termes définie par la signature Σ_{list}, et on utilise le développement précédent pour fournir une décomposition de la sémantique profonde $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\}$.*

On a $Q_{cons}(cons(lst(l_1), l_2)) = \{(lst(l_1), \bot), (\bot, l_2)\}$ et $Q_{nil}(cons(lst(l_1), l_2)) = \{()\}$. De plus, $[\![l_{\mathsf{List}}^{-p_{flat}} \setminus l_2]\!]$ est clairement vide, alors que $[\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!]$ n'est pas vide, puisque $int(z) \in$

$[\![x^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)]\!]$. *Le développement nous donne donc* $\{\!|l^{-p_{flat}}_{\mathsf{List}}|\!\} = [\![l^{-p_{flat}}_{\mathsf{List}}]\!] \cup \{\!|e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)|\!\} \cup \{\!|l^{-p_{flat}}_{\mathsf{List}}|\!\}$. *De même, on a* $\{\!|e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)|\!\} = [\![e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)]\!] \cup \{\!|i_{\mathsf{Int}}|\!\}$ *et* $\{\!|i_{\mathsf{Int}}|\!\} = [\![i_{\mathsf{Int}}]\!] \cup \{\!|i_{\mathsf{Int}}|\!\}$.

On voit que dans la décomposition de $\{\!|l^{-p_{flat}}_{\mathsf{List}}|\!\}$ *et dans celle de* $\{\!|i_{\mathsf{Int}}|\!\}$, *les termes* $\{\!|l^{-p_{flat}}_{\mathsf{List}}|\!\}$ *et* $\{\!|i_{\mathsf{Int}}|\!\}$ *réapparaissent, exprimant l'existence de tels sous-termes. Par induction sur la taille des termes de* $[\![l^{-p_{flat}}_{\mathsf{List}}]\!]$, *respectivement* $[\![i_{\mathsf{Int}}]\!]$, *en remarquant que leurs sous-termes directs sont des termes de* $[\![e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)]\!]$ *ou* $[\![l^{-p_{flat}}_{\mathsf{List}}]\!]$, *respectivement* $[\![i_{\mathsf{Int}}]\!]$, *on obtient donc :*

$$\{\!|l^{-p_{flat}}_{\mathsf{List}}|\!\} = [\![l^{-p_{flat}}_{\mathsf{List}}]\!] \cup [\![e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}]\!]$$

Pour fournir une méthode d'analyse statique, on pourrait facilement automatiser le calcul d'un tel point fixe. Il est alors nécessaire de proposer une méthode systématique pour calculer le développement 4.3 : pour obtenir ce développement, il reste à calculer $Q_c^{-p}(r)$. Étant donné que l'on sait déjà calculer $Q_c(r)$ via l'Algorithme 4.2, il faudrait maintenant proposer une méthode permettant de vérifier de façon systématique que la sémantique close d'un terme de la forme $x_s^{-p} \setminus r$ est vide ou non. On propose, pour se faire, d'essayer de construire un terme de cette sémantique via un graphe décrivant toutes les formes possibles d'un tel terme.

4.1.2 Graphe d'atteignabilité par Exemption

Une méthode facile pour décrire les termes d'une sorte donnée est de construire le graphe décrit par les sortes et les constructeurs de la signature considérée, de façon assez similaire au langage filtré par un automate d'arbre. On peut facilement construire un tel graphe en considérant deux types de nœuds : les nœuds sortes et les nœuds constructeurs. Naturellement, pour un nœud sorte $s \in \mathcal{S}$, on a des arrêtes le connectant aux nœuds constructeurs de $\mathcal{C}_s$, et pour un nœud constructeur $c \in \mathcal{C}$, on a des arrêtes le connectant aux nœuds de son domaine $\mathcal{D}om(c)$.

Exemple 4.4. *On considère l'algèbre de termes définie par la signature* Σ_{list}, *et on construit le graphe comme décrit ci-dessus :*

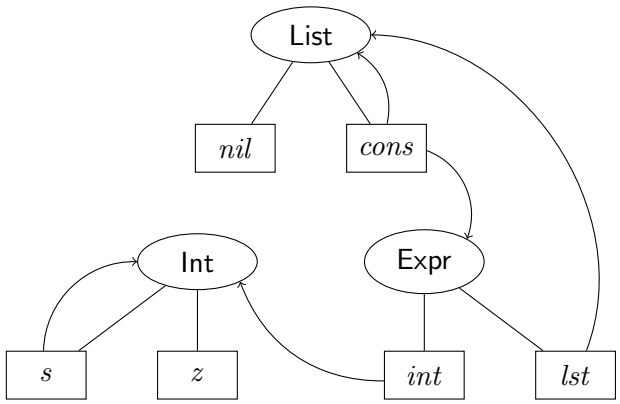


FIGURE 4.3 – Graphe décrivant l'algèbre de termes définie par Σ_{list}, les nœuds constructeurs sont en rectangle et les nœuds sortes en ovale.

Ce graphe permet assez facilement de construire n'importe quelle valeur d'une sorte donnée. Par exemple, pour construire un terme de sorte List, *on part du nœud sorte correspondant et on a deux arrêtes, une pour chaque constructeur de la sorte : nil et cons. On peut donc construire notre terme via n'importe lequel des deux. Si on choisit nil, on obtient la constante nil puisque*

le domaine de nil est vide, noté par l'absence d'arête sortant du nœud nil. Si on choisit cons, il faut instancier le constructeur avec deux sous-termes, un de sorte Expr *et un de sorte* List*. On répète alors l'opération jusqu'à obtenir le terme voulu.*

Dans notre contexte, on ne travaille cependant pas uniquement avec les sortes et les constructeurs donnés par la signature, il faut également considérer des notions de filtrage par motif. En effet, on cherche ici à construire une valeur de la sémantique close d'un terme de la forme $[\![x_s^{-p} \backslash r]\!]$, *i.e.* une valeur de sorte s exempte de p et n'étant pas filtrée par r. Pour cela, on propose donc dans un premier temps de reprendre le formalisme de graphe précédent en l'explicitant en termes de motifs : un nœud sorte s devient un nœud variable x_s et un nœud constructeur $c \in \mathcal{C}$ avec $\mathcal{D}om(c) = s_1 * \cdots * s_n$ devient un nœud motif $c(z_{1_{s_1}}, \ldots, z_{n_{s_n}})$, et pour chaque variable d'un tel nœud, on pointe vers un nœud variable de la même sorte.

Exemple 4.5. *On reprend le graphe présenté dans la Figure 4.3, qui permet de décrire les valeurs de l'algèbre de termes définie par la signature Σ_{list}, avec un formalisme de motif :*

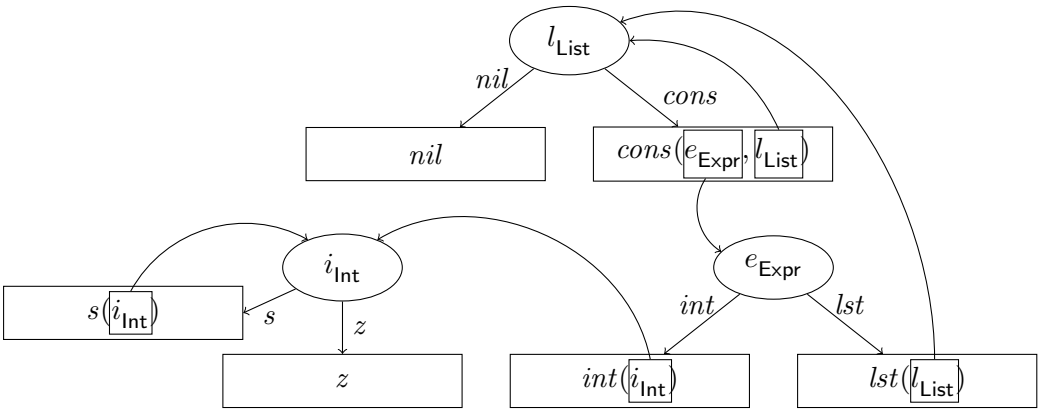


FIGURE 4.4 – Graphe décrivant l'algèbre de termes définie par Σ_{list}, les nœuds motifs sont en rectangle et les nœuds variables en ovale.

Ce graphe peut être utilisé de façon identique au précédent. On peut cependant remarquer que l'approche s'étend naturellement à la sémantique close : pour chaque nœud de ce graphe, on peut construire une valeur de la sémantique close du motif ou de la variable considéré(e).

Ce formalisme de graphe s'étend naturellement à tout terme t de $\mathcal{T}(\mathcal{C}, \mathcal{X})$, de façon à pouvoir facilement construire une valeur de la sémantique close de t. Cela reste encore insuffisant pour résoudre le problème posé ici, qui est de vérifier l'existence d'une valeur dans la sémantique d'un terme de la forme $x_s^{-p} \setminus r$. On observe cependant, qu'en termes de sémantique, les arrêtes entre un nœud variable x_s et ses nœuds motifs $c(z_{1_{s_1}}, \ldots, z_{n_{s_n}})$, avec $c \in \mathcal{C}_s$, se traduit par l'égalité $[\![x_s^{-p}]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(z_{1_{s_1}}, \ldots, z_{n_{s_n}})]\!]$, comme prouvé dans la Proposition 3.5.

De même, pour une (quasi-)variable $x_s^{-p} \backslash r$, on a, avec l'Équation 4.1 et la fonction `computeQc` présentée en Figure 4.2 :

$$[\![x_s^{-p} \setminus r]\!] \quad = \quad \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c(r+p)} [\![c(z_{1s_1}^{-p} \setminus q_1, \ldots, \ldots, z_{ns_n}^{-p} \setminus q_n)]\!] \tag{4.4}$$

On peut donc à partir de l'approche précédente construire un graphe similaire.

Exemple 4.6. *Dans l'algèbre de termes définie par la signature Σ_{list}, on cherche à décrire les valeurs de $[\![l_{\mathsf{List}}^{-p_{flat}}]\!]$, i.e. les valeurs de sorte* List*, exemptes de $p_{flat} = cons(lst(l_1), l_2)$:*

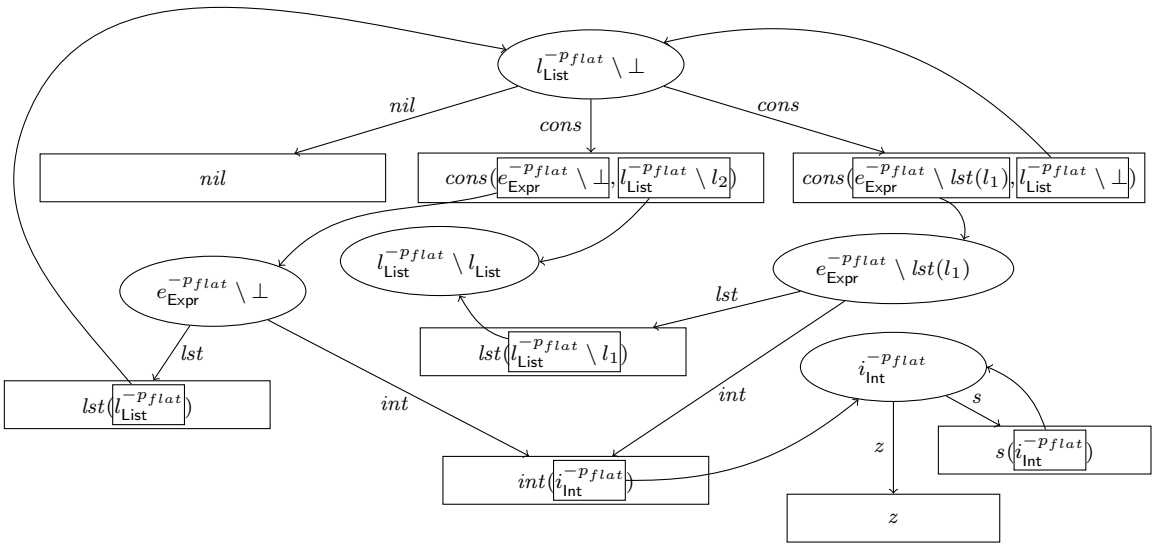


FIGURE 4.5 – Graphe décrivant les termes de $[\![l_{\mathsf{List}}^{-p_{flat}}]\!]$ dans l'algèbre définie par Σ_{list}, les nœuds motifs sont en rectangle et les nœuds variables en ovale.

On peut remarquer, pour toute sorte $s \in \mathcal{S}$, et tous motifs réguliers additifs p et r, que dans le graphe ainsi construit pour décrire les valeurs de la sémantique close de $x_s^{-p} \setminus r$, tous les nœuds (quasi-)variable sont de la forme $z_{s'}^{-p} \setminus r'$ avec des sortes s' et des motifs compléments r' différents mais le même motif annotant p. En effet, cette annotation provient du nœud originellement considéré et est propagé sur les variables fraiches créées par l'Équation 4.4.

De plus, on peut observer qu'un graphe ainsi construit est bien fini, puisque le nombre de nœud (quasi-)variable $z_{s'}^{-p} \setminus r'$ est également fini : on a, en effet, un nombre fini de sortes et, d'après l'algorithme présenté en Figure 4.2, r' est une somme de sous-termes de r et de p, dont il existe aussi seulement un nombre fini (et par construction, il y a un nombre fini de nœud motif pour chaque nœud variable).

On définit formellement ce type de graphe comme suit :

Définition 4.1 (Graphe total)**.** *Étant donné une sorte $s \in \mathcal{S}$, et des motifs réguliers additifs p et r, on appelle* graphe total *de $x_s^{-p} \setminus r$, le couple $G = (V, E)$ où V est un ensemble de nœuds, identifiés par un motif quasi-symbolique, avec $x_s^{-p} \setminus r \in V$, et E est un ensemble d'arêtes orientées, tel que pour tout $u \in V$:*

- *si $u = z_{s'}^{-p} \setminus q$, pour tout $c : s_1 * \cdots * s_n \mapsto s'$ et $(q_1, \ldots, q_n) \in Q_c(p + q)$, $v = c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p} \setminus q_n) \in V$ et $(u, v) \in E$;*
- *si $u = c(u_1, \ldots, u_n)$, pour tout $t \in [1, n]$, $u_i \in V$ et $(u, u_i) \in E$.*

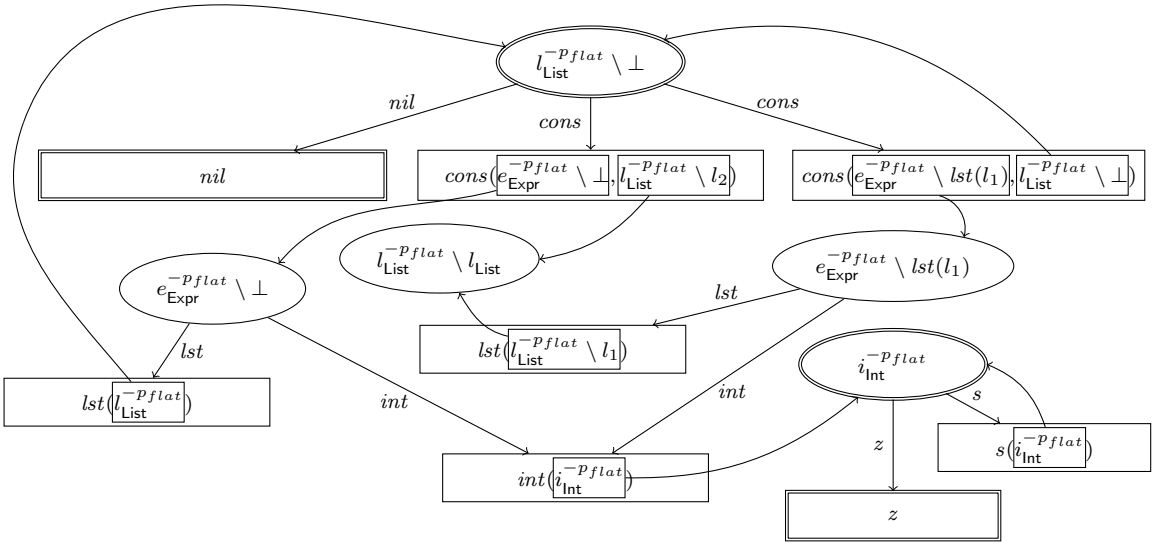
Cependant, certains nœuds du graphe ainsi construit peuvent avoir une sémantique vide. En partant des nœuds variables ayant au moins un nœud motif étant une constante, il faut donc vérifier pour chaque nœud qu'il est possible de construire une valeur finie filtrée par ce nœud.

Définition 4.2 (Nœud instanciable)**.** *Étant donné une sorte $s \in \mathcal{S}$, et des motifs réguliers additifs p et r, on considère un graphe total $G = (V, E)$ de $x_s^{-p} \setminus r \in V$, et pour chaque nœud $u \in V$:*

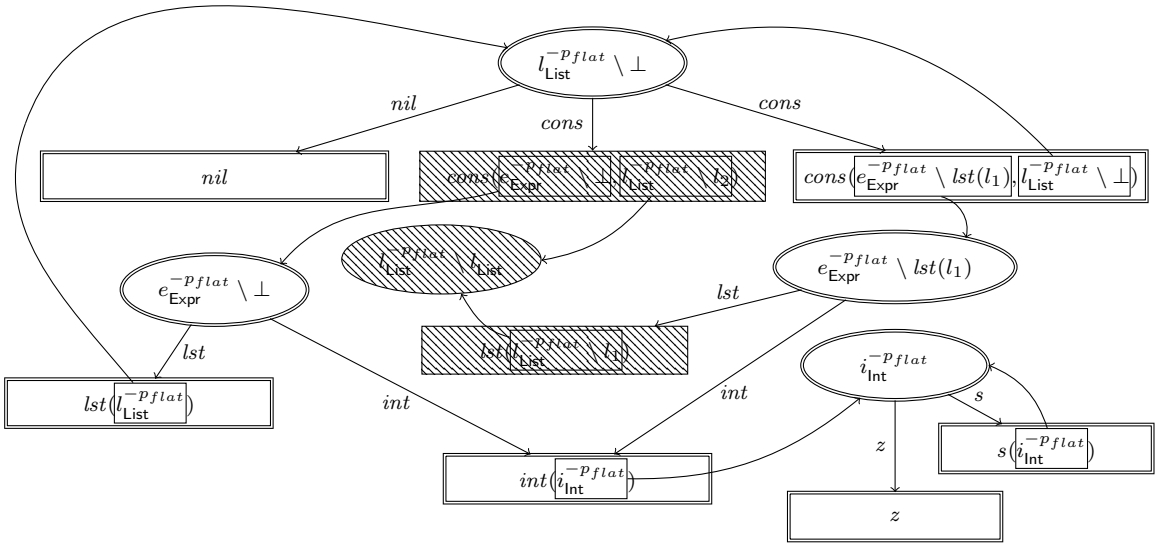
- *si $u = z_{s'}^{-p} \setminus q$, on dit que u est* instanciable *si et seulement il existe $v \in V$* instanciable*, tel que $(u, v) \in E$;*
- *si $u = c(u_1, \ldots, u_n)$, on dit que u est* instanciable *si et seulement si, pour tout $t \in [1, n]$, $u_i \in V$ est* instanciable*.*

Par construction, on voit notamment que tous les nœuds constantes $u = c$ avec $c \in \mathcal{C}^0$ sont instanciables.

Exemple 4.7. *On reprend le graphe présenté en Figure 4.5 et on identifie progressivement les nœuds instanciables :*



Naturellement, on observe dans un premier temps que les nœuds constantes nil, respectivement z, sont instanciables et que les nœuds variables associées $l_{\mathsf{List}}^{-p_{flat}} \setminus \bot$, respectivement $i_{\mathsf{Int}}^{-p_{flat}}$, sont donc également instanciables. En pratique, cela revient à remarquer, qu'il existe bien des valeurs dans la sémantique de ces nœuds. Par la suite, on voit donc que les nœuds $s(i_{\mathsf{Int}}^{-p_{flat}}), int(i_{\mathsf{Int}}^{-p_{flat}}), lst(l_{\mathsf{List}}^{-p_{flat}})$ sont instanciables, et par extension les nœuds variables $e_{\mathsf{Expr}}^{-p_{flat}}$ et $e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)$ aussi. De même, on voit bien que les sémantiques de ces nœuds sont non-vides, puisqu'à partir d'une valeur de la sémantique des nœuds instanciables connus, on peut en effet construire une valeur de la sémantique de ces nœuds. On poursuit le même raisonnement pour vérifier que le nœud $cons(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1), l_{\mathsf{List}}^{-p_{flat}} \setminus \bot)$ est instanciable, et donc sa sémantique non-vide :



Par ce raisonnement, on peut construire, pour chaque nœud instanciable, une valeur de cette sémantique. Inversement, il n'est pas possible de contruire une valeur de la sémantique des nœuds restants, $cons(e_{\mathsf{Expr}}^{-p_{flat}} \setminus \bot, l_{\mathsf{List}}^{-p_{flat}} \setminus l_2)$, $l_{\mathsf{List}}^{-p_{flat}} \setminus l_{\mathsf{List}}$, $lst(l_{\mathsf{List}}^{-p_{flat}} \setminus l_1)$, *et on peut donc conclure que leurs sémantiques sont vides.*

On garde alors uniquement les nœuds instanciables, et on élimine également tous les nœuds qui ne sont plus atteints depuis le nœud d'origine $l_{\mathsf{List}}^{-p_{flat}} \setminus \bot$ *(i.e. le nœud de* $e_{\mathsf{Expr}}^{-p_{flat}} \setminus \bot$ *et ses descendants). On obtient ainsi le graphe suivant :*

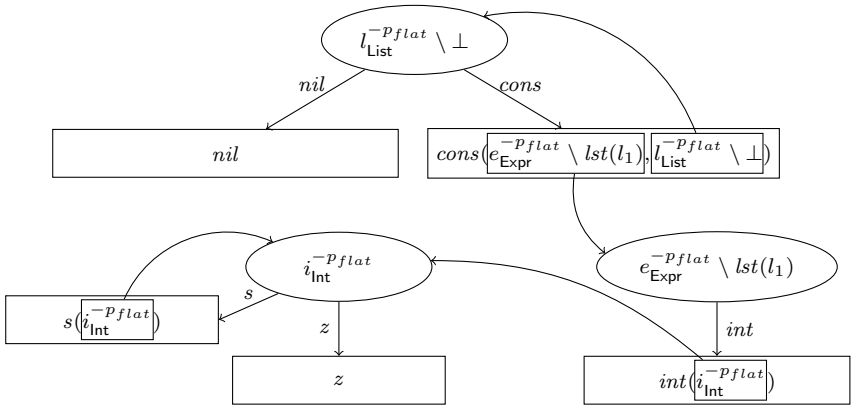


FIGURE 4.6 – Graphe d'atteignabilité de $l_{\mathsf{List}}^{-p_{flat}}$ dans l'algèbre définie par Σ_{list}

On peut enfin utiliser ce graphe pour construire n'importe quelle valeur de la sémantique d'un des nœuds qui le composent. Par exemple, pour construire un terme de $[\![l_{\mathsf{List}}^{-p_{flat}} \setminus \bot]\!]$, *on part du nœud variable correspondant et on a deux arrêtes, une vers le motif nil et* $cons(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1), l_{\mathsf{List}}^{-p_{flat}} \setminus \bot)$. *On peut donc construire notre terme via n'importe lequel des deux. Si on choisit nil, on obtient la constante nil, alors que si on choisit* $cons(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1), l_{\mathsf{List}}^{-p_{flat}} \setminus \bot)$, *il faut instancier les deux variables en répétant l'opération depuis les nœuds variables correspondants. On répète alors l'opération jusqu'à obtenir le terme voulu. On peut ainsi construire un terme de la sémantique de tous les motifs, y compris les (quasi)-variables, composant le graphe, et ainsi*

vérifier que leurs sémantiques ne sont pas vides.

On appelle les graphes ainsi obtenus graphes d'atteignabilité par exemption de motif, que l'on définit formellement comme suit :

Définition 4.3 (Graphe d'atteignabilité)**.** *Étant donné une sorte $s \in \mathcal{S}$, et des motifs réguliers additifs p et r, on appelle graphe d'atteignabilité de $x_s^{-p} \setminus r \in V$ le sous graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ d'un graphe total $G = (V, E)$ de $x_s^{-p} \setminus r \in V$, tel que $\mathcal{V} = \{u \in V \mid u$ est* instanciable*$\}$ et $\mathcal{E} = \{(u, v) \in E \mid u, v \in \mathcal{V}\}$.*

Par construction, on peut utiliser ce type de graphe pour expliciter n'importe quel terme de la sémantique d'un des nœuds qui le compose. L'existence d'un tel terme garantit donc que la sémantique des nœuds est non-vide :

Theorem 4.3. *Étant donné une sorte $s \in \mathcal{S}$, et des motifs additifs réguliers et linéaires p et r, on note $G = (V, E)$ un graphe total de $x_s^{-p} \setminus r$, et $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ le graphe d'atteignabilité obtenu depuis G. On a, pour tout $u \in V$, $[\![u]\!] \neq \emptyset$ si et seulement si $u \in \mathcal{V}$.*

Démonstration. Soient une sorte $s \in \mathcal{S}$, et des motifs additifs réguliers et linéaires p et r, on note $G = (V, E)$ un graphe total de $x_s^{-p} \setminus r$, et $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ le graphe d'atteignabilité obtenu depuis G.

Pour prouver le Théorème, on introduit les notions suivantes :

- Étant donné un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on appelle profondeur de t, notée $d(t)$ la longueur de la plus grande position $\omega \in \mathcal{P}os(t)$;
- Étant donné un nœud $u \in V$, on appelle la distance à feuille de u, notée $dist(u)$, la valeur entière définie par :
  - $dist(u) = 0$, si u est une feuille, *i.e.* $\{(u, v) \in E \mid v \in V\} = \emptyset$;
  - $dist(u) = max_v(d(v)) + 1$ avec $v \in V$ tel que $(u, v) \in E$, si u est un nœud motif tel que $\{(u, v) \in E \mid v \in V\} \neq \emptyset$;
  - $dist(u) = min_v(d(v))$ avec $v \in V$ instanciable tel que $(u, v) \in E$, si u est un nœud (quasi-)variable instanciable tel que $\{(u, v) \in E \mid v \in V\} \neq \emptyset$;
  - $dist(u) = min_v(d(v))$ avec $v \in V$ tel que $(u, v) \in E$, si u est un nœud (quasi-)variable non-instanciable tel que $\{(u, v) \in E \mid v \in V\} \neq \emptyset$.

On prouve en Annexe A le Lemme suivant :

Lemme 4.4. *Étant donné $G = (V, E)$ un graphe total, pour tout nœud $u \in V$, il existe $v \in [\![u]\!]$ avec $d(v) \leq n$ si et seulement u est instanciable avec $dist(u) \leq n$.*

Par définition des nœuds instanciables et de la distance à feuille, tout nœud instanciable a une distance à feuille finie, donc, pour tout $u \in \mathcal{V}$, u est instanciable et il existe donc $v \in [\![u]\!]$ avec $d(v) \geq dist(u)$. Par conséquent, $[\![u]\!] \neq \emptyset$. Inversement, si $[\![u]\!] \neq \emptyset$, alors il existe $v \in [\![u]\!]$ et donc u est instanciable. □

Enfin, on peut observer que la construction de ce graphe et le calcul du point fixe explicité dans la Section 4.1.1 ont une démarche très similaire : dans les deux cas, on part de la (quasi-) variable considérée, on observe les termes générés par la somme explicitée en Équation 4.4, en utilisant la fonction `computeQc` présentée en Figure 4.2, et on répète l'opération pour tous les sous-termes, via la Proposition 4.1. On propose donc de combiner les deux approches pour définir un algorithme permettant de vérifier si la sémantique profonde d'une (quasi-)variable est vide, et sinon d'en donner une décomposition en une union de sémantiques closes.

4.1.3 Algorithme de calcul de la sémantique profonde d'une variable

Dans la Section 4.1.1, on a proposé de décomposer la sémantique profonde d'une (quasi-) variable en calculant le point fixe obtenu via le développement 4.3. Ce développement se repose principalement sur l'Équation 4.2 et la Proposition 4.1, mais dépend du fait que l'on soit capable de vérifier, au préalable, que la sémantique d'une telle variable soit non-vide.

Dans un deuxième temps, on a vu que l'on pouvait également se servir de l'Équation 4.4 et de la Proposition 4.1 pour construire un graphe d'atteignabilité de la (quasi-)variable dont l'on sait que l'ensemble des nœuds sont étiquetés par des motifs quasi-symboliques dont la sémantique est non-vide. On propose donc de montrer que le graphe ainsi obtenu permet directement de calculer une décomposition de la sémantique profonde d'une (quasi-)variable :

Theorem 4.5. *Étant donné une sorte $s \in \mathcal{S}$, des motifs réguliers additifs p et r, et $G = (V, E)$ un graphe d'atteignabilité de $u = x_s^{-p} \setminus r$, on a :*

$$\{\!\{u\}\!\} \quad = \quad \bigcup_{v \in V_u} [\![v]\!]$$

avec V_u l'ensemble des nœuds (quasi-)variables atteignables depuis u.

Démonstration. Soient une sorte $s \in \mathcal{S}$, des motifs réguliers additifs p et r et $G = (V, E)$ un graphe d'atteignabilité de $u = x_s^{-p} \setminus r$.

Soit $v \in V$ un nœud atteignable depuis u dans G. On montre par induction sur le plus court chemin de u à v que $\{\!\{v\}\!\} \subseteq \{\!\{u\}\!\}$.

- Si le chemin est vide, $v = u$ et la preuve est immédiate.
- On suppose maintenant que le chemin est de longueur $n + 1$ avec $n \geq 0$. Il existe donc un nœud $w \in V$ tel que le chemin le plus court de u à w est de longueur n et $(w, v) \in E$. Par induction, on a $\{\!\{w\}\!\} \subseteq \{\!\{u\}\!\}$. De plus, d'après le Théorème 4.3, $\{\!\{v\}\!\} \neq \emptyset$. D'après le développement 4.3, par construction du graphe total, on a donc $\{\!\{v\}\!\} \subseteq \{\!\{w\}\!\}$.

De plus, pour tout motif v on a $[\![v]\!] \subseteq \{\!\{v\}\!\}$, d'où pour tout $v \in V_u$, $[\![v]\!] \subseteq \{\!\{u\}\!\}$.

Soit $w \in \{\!\{u\}\!\}$, par définition de la sémantique profonde, il existe $w' \in [\![u]\!]$ et une position $\omega \in \mathcal{P}os(w')$ tels que $w = w'_{|\omega}$. On montre par induction sur la position ω qu'il existe un nœud (quasi)-variable $v \in V_u$ tel que $w \in [\![v]\!]$.

- Si $\omega = \epsilon$, on a $w = w'_{|\epsilon} = w'$, d'où $w \in [\![u]\!]$.
- Sinon, on a $\omega = \omega'.i$ avec $w'_{|\omega'} = c(w_1, \ldots, w_n)$ et $i \in [1, n]$ tel que $w = w_i$. Par induction, il existe un nœud variable $x_{s'}^{-p} \setminus r' \in V_u$ tel que $w'_{|\omega'} \in [\![x_{s'}^{-p} \setminus r']\!]$. D'après les Équations 4.1 et 4.2, il existe $(q_1, \ldots, q_n) \in Q_c(r' + p)$ tel que $w'_{|\omega'} \in [\![c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p} \setminus q_n)]\!]$. On a donc, pour tout $i \in [1, n]$, $w_i \in [\![z_i s_i^{-p} \setminus q_i]\!]$. D'après le Lemme 4.4, les nœuds $c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p} \setminus q_n)$, $x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n$ sont instanciables, et par construction du graphe d'atteignabilité, ils sont donc atteignables depuis le nœud $x_{s'}^{-p} \setminus r'$, donc depuis u. On a donc bien $w \in [\![x_{s_i}^{-p} \setminus q_i]\!]$ avec $x_{s_i}^{-p} \setminus q_i$ atteignables depuis u.

On a donc bien $\{\!\{u\}\!\} = \bigcup_{v \in V_u} [\![v]\!]$. □

En se basant sur ce Théorème, on peut obtenir la décomposition souhaitée via l'algorithme `getReachable` présenté dans la Figure 4.7.

L'algorithme prend en entrée une sorte s, des motifs additifs réguliers p et r. Il fonctionne en suivant les grandes lignes de la logique présentée dans la Section 4.1.2 :

- on construit le graphe total de $x_s^{-p} \setminus r$ via la fonction `getTotalReach` ;

- on identifie dans ce graphe les nœuds instanciables pour obtenir le graphe d'atteignabilité ;
- on récupère les nœuds atteignables depuis $x_s^{-p} \setminus r$ via la fonction `computeReach`.

Par simplicité, les nœuds variables $x_{s'}^{-p} \setminus r'$ sont représentés par des couples (s', r') et les nœuds motifs $c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p} \setminus q_n)$ sont ignorés et représentés via des arrêtes multiples de la forme $((s', r'), ((s_1, q_1), \ldots, (s_n, q_n)))$.

Les Théorèmes 4.3 et 4.5 garantissent la correction de l'algorithme : le résultat est vide si et seulement si la sémantique considérée est vide, et sinon il en fournit la décomposition souhaitée.

Theorem 4.6. *Étant donné une sorte $s \in \mathcal{S}$, et des motifs réguliers additifs p et r, on note $R =$ `getReachable`(s, p, r) et on a alors :*

- $R = \emptyset$ *si et seulement* $[\![x_s^{-p} \setminus r]\!] = \emptyset$
- $\{\!\{x_s^{-p} \setminus r\}\!\} = \bigcup_{(s',r') \in R} [\![x_{s'}^{-p} \setminus r']\!]$

Démonstration. Soient $s \in \mathcal{S}$, des motifs réguliers additifs p et r.

On note $G = (V, E)$ le graphe total construit depuis $x_s^{-p} \setminus r$, et on considère un ensemble V_0 de nœuds (quasi-)variables de la forme (s', r') et un ensemble E_0 d'arrêtes multiples de la forme $((s', r'), ((s_1, q_1), \ldots, (s_n, q_n)))$ avec $(s', r'), (s_1, q_1), \ldots, (s_n, q_n) \in V_0$, vérifiant l'invariant suivant : pour toute arrête $((s', r'), ((s_1, q_1), \ldots, (s_n, q_n))) \in E$, on a $(s', r') \in V_0$, et il existe $c \in \mathcal{C}_{s'}$ tel que $\mathcal{D}om(c) = s_1 * \cdots * s_n$ et $(q_1, \ldots, q_n) \in Q_c(r + p)$, et $i \in [1, n]$ tel que $(s_i, q_i) \in V_0$ ou $(s_i, q_i) = (s, r)$. On montre que `getTotalReach`(s, p, r, V_0, E_0) renvoie le couple (V_f, E_f) vérifiant l'invariant, avec $V_f = A \uplus V_0$ où A est l'ensemble des nœuds (quasi-)variables atteignables depuis $x_s^{-p} \setminus r$ dans G sans passer par un des nœuds de V_0.

- On commence par remarquer que si l'ensemble A est vide, alors $(s, r) \in V_0$, et on a $(V_f, E_f) = (V_0, E_0)$.
- Si A est non-vide, on a au moins $(s, r) \in A$, d'où l'ajout du nœud dans V. Pour chaque nœud $u \in A \setminus \{(s, r)\}$, il existe $c \in \mathcal{C}_s$ avec $\mathcal{D}om(c) = s_1 * \cdots * s_n$, $(q_1, \ldots, q_n) \in Q_c(r + p)$ et $i \in [1, n]$ tels que u est atteignable depuis (s_i, q_i) sans passer par un des nœuds de $V_0 \cup \{(s, r)\}$. Pour chacun de ces nœuds, on appelle donc `getTotalReach` en préservant l'invariant entre les deux derniers arguments.

Pour l'appelle initial, on a $V_0 = \emptyset$ et $E_0 = \emptyset$ d'où V_f est l'ensemble des nœuds variables du graphe total construit depuis $x_s^{-p} \setminus r$ et E_f l'ensemble des arrêtes multiples associé.

Dans `getReachable`, on construit ensuite le graphe d'atteignabilité à partir du graphe total obtenu en recherchant les nœuds variables instanciables. On utilise la notion de distance à feuille d'un nœud, introduite dans la preuve du Théorème 4.3. On commence par identifier, les nœuds variables pouvant être instanciés par une constante, *i.e.* ayant une distance à feuille de 0, puis en recherchant dans les nœuds restant les nœuds instanciables de distance à feuille augmentée de 1 à chaque itération. Comme la distance à feuille est bornée par la taille du graphe total, on obtient bien le sous ensemble de nœuds instanciables.

Enfin, `computeReach` effectue une recherche d'atteignabilité dans le graphe total en se limitant aux nœuds instanciables et aux arrêtes multiples où tous les nœuds sont instanciables. Le résultat R correspond donc à l'ensemble $A_{x_s^{-p} \setminus r}$ des nœuds variables instanciables et atteignables depuis $x_s^{-p} \setminus r$. D'après le Théorème 4.3, si $[\![x_s^{-p} \setminus r]\!] = \emptyset$ alors le nœud $x_s^{-p} \setminus r$ n'est pas instanciable et on a donc $R = \emptyset$. Et d'après le Théorème 4.5, on a bien $\{\!\{x_s^{-p} \setminus r\}\!\} = \bigcup_{(s',r') \in R} [\![x_{s'}^{-p} \setminus r']\!]$. □

Le problème initial étant de montrer qu'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ est exempt d'un motif p en vérifiant que $\{\!\{t\}\!\} \cap [\![p]\!] = \emptyset$ (Proposition 3.9), on peut finalement décomposer la sémantique

profonde de t en une union de sémantique close, pour ramener le calcul de l'intersection $\{\!|t|\!\} \cup [\![p]\!]$ en un problème de réduction de conjonction de motifs.

Exemple 4.8. *On considère l'algèbre de termes définie par la signature Σ_{list}, grâce à l'algorithme* `getReachable`*, on peut facilement vérifier que la sémantique de $l_{\mathsf{List}}^{-cons(e,l)} \setminus nil$ est vide puisqu'on a* `getReachable`$(\mathsf{List}, cons(e,l), nil) = \emptyset$.

De plus, pour prouver que $u = cons(int(i), flatten(l))$ est exempt de $p_{flat} = cons(lst(l_1), l_2)$, on peut remarquer que son équivalent sémantique est $\tilde{u} = cons(int(i_{\mathsf{Int}}^{-\bot}), l_{\mathsf{List}}^{-p_{flat}})$, et que comme `getReachable`$(\mathsf{List}, p_{flat}, \bot) = \{(\mathsf{List}, \bot), (\mathsf{Expr}, lst(l_1)), (\mathsf{Int}, \bot)\}$*, on a, avec les Propositions 3.8 et 4.1 :*

$$
\begin{array}{rcl}
\{\!|u|\!\} \cap [\![p_{flat}]\!] & = & \{\!|\tilde{u}|\!\} \cap [\![p_{flat}]\!] \\
 & \overset{Pr\,4.1}{=} & ([\![\tilde{u}]\!] \cup [\![int(i_{\mathsf{Int}}^{-\bot})]\!] \cup [\![i_{\mathsf{Int}}^{-\bot}]\!] \cup \{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\}) \cap [\![p_{flat}]\!] \\
 & \overset{Th\,4.6}{=} & ([\![\tilde{u}]\!] \cup [\![int(i_{\mathsf{Int}}^{-\bot})]\!] \cup [\![i_{\mathsf{Int}}^{-\bot}]\!] \cup [\![l_{\mathsf{List}}^{-p_{flat}} \setminus \bot]\!] \\
 & & \qquad \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}} \setminus \bot]\!]) \cap [\![p_{flat}]\!] \\
 & \overset{Pr\,3.8}{=} & [\![\tilde{u} \times p_{flat}]\!] \cup [\![int(i_{\mathsf{Int}}^{-\bot}) \times p_{flat}]\!] \cup [\![i_{\mathsf{Int}}^{-\bot} \times p_{flat}]\!] \cup [\![(l_{\mathsf{List}}^{-p_{flat}} \setminus \bot) \times p_{flat}]\!] \\
 & & \cup [\![(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)) \times p_{flat}]\!] \cup [\![(i_{\mathsf{Int}}^{-p_{flat}} \setminus \bot) \times p_{flat}]\!]
\end{array}
$$

Afin de montrer que u est exempt de p_{flat}, il reste maintenant à montrer que les sémantiques des conjonctions $\tilde{u} \times p_{flat}$, $int(i_{\mathsf{Int}}^{-\bot}) \times p_{flat}$, $i_{\mathsf{Int}}^{-\bot} \times p_{flat}$, $(l_{\mathsf{List}}^{-p_{flat}} \setminus \bot) \times p_{flat}$, $(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)) \times p_{flat}$ et $(i_{\mathsf{Int}}^{-p_{flat}} \setminus \bot) \times p_{flat}$ sont vides, i.e. qu'elles peuvent être réduites, en conservant leur sémantique, à $\bot$.

On propose donc, dans la Section suivante, un système de réduction, conservant la sémantique close des motifs et permettant de réduire une conjonction $t \times p$ où t est un motif quasi-additif et u additif régulier de telle sorte que la forme normale obtenue est $\bot$ si et seulement la sémantique close de la conjonction est vide.

4.2 Comparaison de motif

Pour construire une méthode systématique de vérification des propriétés d'Exemption de motif, on propose donc de se baser sur l'équivalence donnée par la Proposition 3.9, en vérifiant que l'intersection entre la sémantique profonde du terme considéré et la sémantique close du motif est vide. On a vu dans la Section précédente qu'on pouvait décomposer une sémantique profonde en une union de sémantiques closes, et ainsi ramener le problème à l'évaluation de la sémantique close de conjonctions de motifs. Pour vérifier que ces derniers ont une sémantique vide, on s'inspire, cette fois, de l'approche présentée dans [CM19], qui utilise un système de réécriture préservant la sémantique close des motifs pour réduire ces derniers à une forme normale permettant, notamment, de déterminer si leur sémantique est vide.

4.2.1 Présentation générale

L'approche proposée dans [CM19] fournit une méthode pour étudier un motif étendu de façon à proposer une forme équivalente du motif original. Pour se faire, la forme souhaitée est obtenue par réécriture via un système conservant la sémantique des motifs et garantissant que la forme normale obtenue par réduction d'un motif étendu quelconque soit un motif additif. Une telle forme normale permet notamment de décomposer un complément de motifs en une disjonction

de motifs, tout en garantissant que le filtrage du motif initial soit équivalent au filtrage d'au moins un motif de la disjonction. Par extension, cette approche permet d'identifier les motifs étendus dont la sémantique est vide puisque la forme normale alors obtenue serait donc une disjonction vide, *i.e.* le motif vide $\bot$.

On souhaite donc proposer un système similaire, adapté au formalisme considéré ici. La méthode originale ne s'applique en effet pas directement dans notre cas, puisque notre formalisme considère des motifs étendus définis avec un opérateur explicite de conjonctions de motifs mais également des variables annotées, qui n'existaient pas dans l'approche proposée par [CM19]. Une adaptation directe et complète du système original, n'est cependant pas possible puisque ces constructions supplémentaires ne permettent d'obtenir une forme normale commune à la réduction de tout motif considéré.

Cela dit, on a vu que, pour pouvoir vérifier des propriétés d'Exemption de Motif, la méthode proposée doit seulement être capable de vérifier que la sémantique close d'une conjonction entre un motif quasi-additif t et un motif additif linéaire et régulier p est vide. Par conséquent, plutôt que de proposer un système permettant de réduire tout motif étendu, comme dans [CM19], on peut se restreindre à l'étude des motifs de la forme $t \times p$. En particulier, le système proposé doit être capable de réduire la conjonction $t \times p$ à $\bot$ si et seulement si sa sémantique est vide.

On propose donc un système de réécriture permettant de réduire certaines formes de motifs en préservant la sémantique. Comme dans le système d'origine, les opérateurs de motifs sont équivalents à des opérations ensemblistes (cf. Proposition 3.8) : la sémantique d'une disjonction, respectivement conjonction, respectivement complément, est l'union, respectivement l'intersection, respectivement la différence des sémantiques, et la sémantique d'un motif ayant un symbole constructeur en tête se comporte comme un produit cartésien des sémantiques des sous-termes. On peut donc s'inspirer des lois de simplification des opérations ensemblistes pour écrire les règles du système souhaité.

De plus, comme on l'a vu dans la Section précédente, bien que réduire un motif de la forme $x_s^{-p} \setminus r$ peut être problématique puisque cela introduirait un complément $\setminus p$ supplémentaire, rendant le système non-terminant, on est capable de vérifier, en utilisant l'algorithme `getReachable`, présenté dans la Figure 4.7, si sa sémantique est vide. On peut alors simplement construire un système qui ne réduira un tel motif que dans le cas où sa sémantique est bien vide.

4.2.2 Système $\mathfrak{R}$

On propose ainsi le système $\mathfrak{R}$, présenté dans la Figure 4.8. Comme on souhaite que $\mathfrak{R}$ préserve la sémantique close des motifs, les règles correspondent généralement aux lois de compositions des opérateurs ensemblistes où les motifs constructeurs se comportent comme un produit cartésien, et les opérateurs comme leur opération ensembliste équivalente (voir Proposition 3.8).

Les règles A1 et A2, respectivement E2 et E3, expriment le fait que le motif vide $\bot$ correspond à l'élément neutre de la disjonction, respectivement élément absorbant de la conjonction, tandis que la règle E1 indique que la sémantique d'un motif constructeur dont l'un des sous-terme a une sémantique vide est également vide :

(A1) D'après Proposition 3.8, $[\![\bot + \overline{v}]\!] = [\![\bot]\!] \cup [\![\overline{v}]\!] = [\![\overline{v}]\!]$.

(A2) D'après Proposition 3.8, $[\![\overline{v} + \bot]\!] = [\![\overline{v}]\!] \cup [\![\bot]\!] = [\![\overline{v}]\!]$.

(E1) D'après Proposition 3.8,
$[\![\delta(\overline{v_1}, \ldots, \bot, \ldots, \overline{v_n})]\!] = \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![\overline{v_1}]\!] \times \cdots \times \emptyset \times \cdots \times [\![\overline{v_n}]\!]\} = \emptyset$.

(E2) D'après Proposition 3.8, $[\![\bot \times \overline{v}]\!] = [\![\bot]\!] \cap [\![\overline{v}]\!] = \emptyset$.

(E3) D'après Proposition 3.8, $[\![\overline{v} \times \bot]\!] = [\![\overline{v}]\!] \cap [\![\bot]\!] = \emptyset$.

La règle S1 exprime la distributivité de produit cartésien sur la disjonction, tandis que les règles S2 et S3 expriment la distributivité de la conjonction sur la disjonction. Enfin, la règle S4 correspond à l'associativité de la disjonction.

(S1) D'après Proposition 3.8,
$$\begin{aligned}\llbracket\delta(\overline{v_1},\dots,\overline{v_i}+\overline{w_i},\dots,\overline{v_n})\rrbracket &= \{\delta(t_1,\dots,t_n) \mid (t_1,\dots,t_n) \in \llbracket\overline{v_1}\rrbracket\times\cdots\times\llbracket\overline{v_i}+\overline{w_i}\rrbracket\times\cdots\times\llbracket\overline{v_n}\rrbracket\}\\ &= \llbracket\delta(\overline{v_1},\dots,\overline{v_i},\dots,\overline{v_n}) + \delta(\overline{v_1},\dots,\overline{w_i},\dots,\overline{v_n})\rrbracket\end{aligned}$$

(S2) D'après Proposition 3.8, $\llbracket(\overline{v_1}+\overline{v_2})\times\overline{w}\rrbracket = (\llbracket\overline{v_1}\rrbracket\cup\llbracket\overline{v_2}\rrbracket)\cap\llbracket\overline{w}\rrbracket = (\llbracket\overline{v_1}\rrbracket\cap\llbracket\overline{w}\rrbracket)\cup(\llbracket\overline{v_2}\rrbracket\cap\llbracket\overline{w}\rrbracket) = \llbracket(\overline{v_1}\times\overline{w})+(\overline{v_2}\times\overline{w})\rrbracket$.

(S3) D'après Proposition 3.8, $\llbracket\overline{v}\times(\overline{w_1}+\overline{w_2})\rrbracket = \llbracket\overline{v}\rrbracket\cap(\llbracket\overline{w_1}\rrbracket\cup\llbracket\overline{w_2}\rrbracket) = (\llbracket\overline{v}\rrbracket\cap\llbracket\overline{w_1}\rrbracket)\cup(\llbracket\overline{v}\rrbracket\cap\llbracket\overline{w_2}\rrbracket) = \llbracket(\overline{v}\times\overline{w_1})+(\overline{v}\times\overline{w_2})\rrbracket$.

(S4) D'après Proposition 3.8, $\llbracket\overline{u}+(\overline{v}+\overline{w})\rrbracket = \llbracket\overline{u}\rrbracket\cup(\llbracket\overline{v}\rrbracket\cup\llbracket\overline{w}\rrbracket) = (\llbracket\overline{u}\rrbracket\cup\llbracket\overline{v}\rrbracket)\cup\llbracket\overline{w}\rrbracket = \llbracket(\overline{u}+\overline{v})+\overline{w}\rrbracket$.

Les règles M1–M6' expriment les propriétés classiques de la différence, induites par les lois de compositions de la différence en théorie des ensembles :

(M1) Par définition de la sémantique close, on a $\llbracket\overline{x}_s^{-\bot}\rrbracket = \mathcal{T}_s(\mathcal{C},\mathcal{X})$, d'où pour tout motif $\overline{v} : s$, $\llbracket\overline{v}\rrbracket \subseteq \llbracket\overline{x}_s^{-\bot}\rrbracket$. Donc, d'après Proposition 3.8, $\llbracket\overline{v}\setminus\overline{x}_s^{-\bot}\rrbracket = \llbracket\overline{v}\rrbracket\setminus\llbracket\overline{x}_s^{-\bot}\rrbracket = \emptyset$.

(M2) D'après Proposition 3.8, $\llbracket\overline{v}\setminus\bot\rrbracket = \llbracket\overline{v}\rrbracket\setminus\llbracket\bot\rrbracket = \llbracket\overline{v}\rrbracket$.

(M3') D'après Proposition 3.8, $\llbracket(\overline{v_1}+\overline{v_2})\setminus\overline{w}\rrbracket = (\llbracket\overline{v_1}\rrbracket\cup\llbracket\overline{v_2}\rrbracket)\setminus\llbracket\overline{w}\rrbracket = (\llbracket\overline{v_1}\rrbracket\setminus\llbracket\overline{w}\rrbracket)\cup(\llbracket\overline{v_2}\rrbracket\setminus\llbracket\overline{w}\rrbracket) = \llbracket(\overline{v_1}\setminus\overline{w})+(\overline{v_2}\setminus\overline{w})\rrbracket$.

(M5) D'après Proposition 3.8, $\llbracket\bot\setminus\overline{v}\rrbracket = \llbracket\bot\rrbracket\setminus\llbracket\overline{v}\rrbracket = \emptyset$.

(M6') D'après Proposition 3.8, $\llbracket\alpha(\overline{v_1},\dots,\overline{v_n})\setminus(\overline{v}+\overline{w})\rrbracket = \llbracket\alpha(\overline{v_1},\dots,\overline{v_n})\rrbracket\setminus(\llbracket\overline{v}\rrbracket\cup\llbracket\overline{w}\rrbracket) = (\llbracket\alpha(\overline{v_1},\dots,\overline{v_n})\rrbracket\setminus\llbracket\overline{v}\rrbracket)\setminus\llbracket\overline{w}\rrbracket = \llbracket(\alpha(\overline{v_1},\dots,\overline{v_n})\setminus\overline{v})\setminus\overline{w}\rrbracket$

Les règles M7 et M8 correspondent à la distributivité de la différence sur un produit cartésien étiqueté par le symbole de tête des motifs : M7 quand le symbole est le même et les produits cartésiens sont comparables, M8 quand les produits cartésiens sont disjoints. Ces règles découlent directement du Lemme 4.2, prouvé dans la Section précédente. De plus, quand l'arité du symbole α est 0, la somme dans le membre droit de M7 est vide et on a donc $\alpha\setminus\alpha \twoheadrightarrow \bot$.

Le système proposé dans [CM19] possède une règle M4 pour décrire le cas complément sur une variable : $\overline{x}_s\setminus\alpha(\overline{v_1},\dots,\overline{v_n}) \twoheadrightarrow \left(\sum_{c\in\mathcal{C}_s} c(z_{1s_1},\dots,z_{ns_n})\right)\setminus\alpha(\overline{v_1},\dots,\overline{v_n})$. Une règle correspondante dans notre formalisme devrait cependant prendre en compte l'annotation de la variable, et d'après l'Équation 4.1, serait donc de la forme : $\overline{x}_s^{-\overline{p}}\setminus\alpha(\overline{v_1},\dots,\overline{v_n}) \twoheadrightarrow \left(\sum_{c\in\mathcal{C}_s} c(z_{1s_1}^{-\overline{p}},\dots,z_{ns_n}^{-\overline{p}})\right)\setminus(\alpha(\overline{v_1},\dots,\overline{v_n})+\overline{p})$. Une telle règle introduirait cependant de nouveaux termes du côté droit de la différence, rendant ainsi le système non-terminant. On dépendra donc de l'algorithme `getReachable` pour vérifier que la sémantique d'une telle différence est vide (cf. règle P7 plus bas).

Les règles T1 et T2 exprime le fait qu'une variable annotée $-\bot$ a pour sémantique l'ensemble des valeurs de sa sorte, et se comporte donc comme l'élément neutre de la conjonction (de même, la conjonction de deux variables annotées de façon identique n'a pas d'effet). On garde cependant l'information concernant le filtrage de la variable par le biais de l'alias :

(T1) Par définition de la sémantique close, on a $\llbracket\overline{y}_s^{-\overline{p}}\rrbracket \subseteq \llbracket\overline{x}_s^{-\bot}\rrbracket = \mathcal{T}_s(\mathcal{C},\mathcal{X})$. D'où, d'après Proposition 3.8, $\llbracket\overline{x}_s^{-\bot}\times\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket\overline{x}_s^{-\bot}\rrbracket\cap\llbracket\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket x \mathbin{@} \overline{y}_s^{-\overline{p}}\rrbracket$.

(T2) De façon similaire, on a $\llbracket\overline{x}_s^{-\overline{p}}\times\overline{y}_s^{-\bot}\rrbracket = \llbracket\overline{x}_s^{-\overline{p}}\rrbracket\cap\llbracket\overline{y}_s^{-\bot}\rrbracket = \llbracket\overline{x}_s^{-\overline{p}}\rrbracket = \llbracket x \mathbin{@} \overline{y}_s^{-\overline{p}}\rrbracket$. Et comme $\llbracket\overline{x}_s^{-\overline{p}}\rrbracket = \llbracket\overline{y}_s^{-\overline{p}}\rrbracket$, $\llbracket\overline{x}_s^{-\overline{p}}\times\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket\overline{x}_s^{-\overline{p}}\rrbracket\cap\llbracket\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket\overline{y}_s^{-\overline{p}}\rrbracket = \llbracket x \mathbin{@} \overline{y}_s^{-\overline{p}}\rrbracket$.

Les règles T3 et T4 expriment la distributivité de la conjonction sur le produit cartésien étiqueté par le symbole de tête : T3 quand le symbole est identique et les produits cartésiens sont comparables, T4 quand les produits cartésiens sont disjoints.

(T3) D'après Proposition 3.8, $[\![\alpha(\overline{v_1},\ldots,\overline{v_n}) \times \alpha(\overline{w_1},\ldots,\overline{w_n})]\!] = \{\alpha(t_1,\ldots,t_n) | (t_1,\ldots,t_n) \in ([\![\overline{v_1}]\!] \cap [\![\overline{w_1}]\!]) \times \cdots \times ([\![\overline{v_n}]\!] \cap [\![\overline{w_n}]\!])\} = [\![\alpha(\overline{v_1} \times \overline{w_1},\ldots,\overline{v_n} \times \overline{w_n})]\!]$.

(T4) D'après Proposition 3.8, $[\![\alpha(\overline{v_1},\ldots,\overline{v_n}) \times \beta(\overline{w_1},\ldots,\overline{w_m})]\!] = \{\alpha(t_1,\ldots,t_n) \mid (t_1,\ldots,t_n) \in [\![\overline{v_1}]\!] \times \cdots \times [\![\overline{v_n}]\!]\} \cap \{\beta(t_1,\ldots,t_m) \mid (t_1,\ldots,t_m) \in [\![\overline{w_1}]\!] \times \cdots \times [\![\overline{w_m}]\!]\} = \emptyset$.

La règle L1 exprime le fait qu'aliasser le motif vide $\bot$ est inutile, et la règle L2 qu'aliasser une somme de motifs revient à aliasser chaque élément de la somme. La règle L3, respectivement les règles L4 et L5, exprime(nt) la propagation de l'alias sur un complément, respectivement sur une conjonction.

(L1) D'après Proposition 3.8, $[\![\overline{x} \mathbin{@} \bot]\!] = \emptyset$

(L2) D'après Proposition 3.8, $[\![\overline{x} \mathbin{@} (\overline{v} + \overline{w})]\!] = [\![\overline{v}]\!] \cup [\![\overline{w}]\!] = [\![\overline{x} \mathbin{@} \overline{v} + \overline{x} \mathbin{@} \overline{w}]\!]$.

(L3) D'après Proposition 3.8, $[\![(\overline{x} \mathbin{@} \overline{v}) \setminus \overline{w}]\!] = [\![\overline{v}]\!] \setminus [\![\overline{w}]\!] = [\![\overline{x} \mathbin{@} (\overline{v} \setminus \overline{w})]\!]$.

(L4) D'après Proposition 3.8, $[\![(\overline{x} \mathbin{@} \overline{v}) \times \overline{w}]\!] = [\![\overline{v}]\!] \cap [\![\overline{w}]\!] = [\![\overline{x} \mathbin{@} (\overline{v} \times \overline{w})]\!]$.

(L5) D'après Proposition 3.8, $[\![\overline{v} \times (\overline{x} \mathbin{@} \overline{w})]\!] = [\![\overline{v}]\!] \cap [\![\overline{w}]\!] = [\![\overline{x} \mathbin{@} (\overline{v} \times \overline{w})]\!]$.

Les règles P1 et P2 découlent de l'observation que, comme prouvé dans la Proposition 3.8, on a $[\![\overline{x}_s^{-\overline{p}}]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(z_{1s_1}^{-\overline{p}},\ldots,z_{ns_i}^{-\overline{p}}) \setminus p]\!]$, et que d'après les règles S2/S3, T4 et A1/A2, une variable $\overline{x}_s^{-\overline{p}}$ en conjonction avec un motif $\alpha(\overline{v_1},\ldots,\overline{v_n})$ se réduit donc à $\alpha(z_{1s_1}^{-\overline{p}},\ldots,z_{ns_i}^{-\overline{p}}) \setminus p$. De plus, comme pour T1/T2, l'information concernant le filtrage de la variable est conservée par le biais d'un alias. On note également que les règles ne s'appliquent que sur un motif bien sorté, donc, pour P1 et P2, tel que $\alpha \in \mathcal{C}_s$; la sémantique d'un motif mal sorté est en effet vide puisqu'il ne pourrait filtré que des termes mal sortés, et un tel motif est donc implicitement réduit à $\bot$. Enfin les variables $z_{1s_1}^{-\overline{p}},\ldots,z_{ns_i}^{-\overline{p}}$ introduites sont des variables fraîches automatiquement générées (et généralement ignorées par les alias du point de vue de l'implémentation).

Comme expliqué précédemment, pour réduire une (quasi-)variable $\overline{x}_s^{-\overline{p}} \setminus \overline{t}$, on dépend de l'algorithme `getReachable` pour vérifier si $[\![\overline{x}_s^{-\overline{p}} \setminus \overline{t}]\!] = \emptyset$. Dans le cas d'une conjonction avec un tel terme, si $[\![\overline{x}_s^{-\overline{p}} \setminus \overline{t}]\!] \neq \emptyset$, on priorise alors la conjonction, comme explicité par les règles P3, P4 et P5. Les trois règles expriment la même loi d'associativité de la conjonction sur le complément, mais en se restreignant sur les cas applicables pour garantir la confluence du système :

$$\begin{array}{rcl} [\![(\overline{v} \times \overline{w}) \setminus \overline{u}]\!] & = & [\![\overline{v}]\!] \cup [\![\overline{w}]\!] \setminus [\![\overline{u}]\!] \\ & = & ([\![\overline{v}]\!] \setminus [\![\overline{u}]\!]) \cup [\![\overline{w}]\!] \quad = \quad [\![(\overline{v} \setminus \overline{u}) \times \overline{w}]\!] \\ & = & [\![\overline{v}]\!] \cup ([\![\overline{w}]\!] \setminus [\![\overline{u}]\!]) \quad = \quad [\![\overline{v} \times (\overline{w} \setminus \overline{u})]\!] \end{array}$$

Enfin, quand `getReachable` vérifie que $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} = \emptyset$, la règle P7 réalise la transformation correspondante en réduisant le motif à $\bot$. De plus, pour garantir la complétude de l'application de P7, il est nécessaire de factoriser les compléments autour des (quasi-)variables, d'où la règle P6 (qui correspond donc à la réciproque de la règle M6', dans le cas des variables).

On peut également noter que la Figure 4.8 définit des schémas de règle et qu'en pratique le système dépend donc de la signature considérée : pour les règles utilisant des symboles α, β, δ, ces derniers sont instanciés par chaque symbole constructeur de $\mathcal{C}$ pour générer le système concret. De plus, les conditions des règles P3–P7 sont vérifiées indépendamment en utilisant l'algorithme `getReachable` présenté dans la Section précédente.

Le système $\mathfrak{R}$, ainsi définit, préserve la plupart des propriétés structurelles des motifs :

Proposition 4.7. *Soient des motifs u et v tels que $u \Longrightarrow_{\mathfrak{R}} v$, on a*

- *si u est régulier, v est régulier ;*

- *si u est additif, v est additif ;*
- *si u est quasi-additif, v est quasi-additif ;*
- *si u est linéaire, v est linéaire.*

Démonstration. Pour prouver la préservation de la régularité, on commence par remarquer qu'aucune règle de $\mathfrak{R}$ modifie l'annotation des variables et les seules règles introduisant de nouvelles variables annotées, *i.e.* P1 et P2, annotent les nouvelles variables avec une annotation du motif de départ, *i.e.* $\perp$ pour des motifs réguliers. De plus, pour tous motifs réguliers t et u, pour toute position $\omega \in \mathcal{P}os(t), t_{|\omega}$ et $t\,[u]_\omega$ sont réguliers. La régularité des motifs est donc préservée par réduction suivant $\mathfrak{R}$.

De même, pour prouver qu'un motif additif est forcément réduit en un motif additif, on observe que les seules règles s'appliquant à un motif additif sont A1, A2, E1, S1, S4, L1 et L2 qui réduisent des motifs additifs en des motifs additifs. De plus, pour tous motifs additifs t et u, pour toute position $\omega \in \mathcal{P}os(t), t_{|\omega}$ et $t\,[u]_\omega$ sont additifs. Les motifs additifs sont donc réduits par $\mathfrak{R}$ en des motifs additifs.

Les seules règles s'appliquant des motifs quasi-additifs sont A1, A2, E1, S1, S4, M1, M2, L1, L2, et P7, qui réduisent tout motif quasi-additif en un motif quasi-additif. On peut aussi remarquer que pour tous motifs quasi-additifs t et u, pour toute position $\omega \in \mathcal{P}os(t)$, $t_{|\omega}$ est un motif quasi-additif et on a :

- si $t_{|\omega}$ et u sont additifs alors, $t\,[u]_\omega$ est quasi-additif ;
- si $t_{|\omega}$ n'est pas additif, $t\,[u]_\omega$ est quasi-additif.

Par conséquent, comme l'additivité des motifs est préservée par réduction suivant $\mathfrak{R}$, $\mathfrak{R}$ réduit un motif quasi-additif en un motif quasi-additif.

Pour prouver la préservation de la linéarité, on commence par observer que pour toutes les règles $ls \rightharpoonup rs \in \mathfrak{R}$, pour toute variable $x \in \mathcal{MV}(rs)$, soit $x \in \mathcal{MV}(ls)$ soit x est une variable fraîche, et réduit un motif linéaire en un motif linéaire. De plus, pour tous motifs linéaires t et u, pour toute position $\omega \in \mathcal{P}os(t), t_{|\omega}$ est linéaire et si toute variable de $\mathcal{MV}(u) \setminus \mathcal{MV}(t_{|\omega})$ est fraîche, alors $t\,[u]_\omega$ est linéaire. Donc la linéarité des motifs est préservée par réduction suivant $\mathfrak{R}$. □

De plus, comme on l'a vu, les règles du système $\mathfrak{R}$ ont été définies à partir de lois de compositions d'opérateurs ensemblistes, de façon à ce que chacune préserve la sémantique close des motifs. On en déduit que le système préserve la sémantique close : [1]

Proposition 4.8 (Préservation de la sémantique)**.** *Soient des motifs u et v, si $u \Longrightarrow_{\mathfrak{R}}^{*} v$ alors $[\![u]\!] = [\![v]\!]$.*

Démonstration. On a montré que toutes les règles du système $\mathfrak{R}$ préservent la sémantique des motifs linéaires. De plus on a pour tous motifs linéaires t, u, pour toute position $\omega \in \mathcal{P}os(t)$, si $[\![t_{|\omega}]\!] = [\![u]\!]$ et tout $x \in \mathcal{MV}(u) \setminus \mathcal{MV}(t_{|\omega})$ est une variable fraîche, alors $[\![t]\!] = [\![t\,[u]_\omega]\!]$. Donc la réduction suivant $\mathfrak{R}$ préserve la sémantique des motifs linéaires. □

De plus le système est confluent et terminant :

Proposition 4.9. *Le système $\mathfrak{R}$ est confluent et terminant.*

1. on montrera que le système $\mathfrak{R}$ préserve également la sémantique des motifs non-linéaires dans le Chapitre suivant

Démonstration. On prouve en Annexe A la confluence locale du système en montrant que chaque paire critique induite des règles de réécriture converge.

Pour la terminaison, on observe simplement que l'ordre lexicographique induit par la priorité $\bot < + < @ < \overline{x} < \alpha < \setminus < \times$ est strictement décroissant pour toutes les règles de $\mathfrak{R}$. On fournit également, en Annexe B, un méta-encodage du système dont la terminaison peut être vérifiée à l'aide d'outils de vérification automatique comme TTT2 ou AProVE. □

On peut donc donner une caractérisation des formes normales obtenues par réduction via la relation induite par $\mathfrak{R}$. Dans notre cas, on s'intéresse principalement à la réduction de conjonction $t \times p$ où t est un motif quasi-additif et p est un motif additif linéaire et régulier. La forme normale obtenue par réduction d'une telle conjonction est soit $\bot$ soit une somme de motifs quasi-symboliques :

Proposition 4.10. *Soient un motif quasi-additif t et un motif additif linéaire et régulier p, $v := (t \times p)\downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t \times p]\!] = \emptyset$.*

Démonstration. Par confluence de $\mathfrak{R}$, on peut étudier les formes normales des différentes combinaisons concernées par la réduction. On prouve ainsi, en Annexe A, le Lemme suivant :

Lemme 4.11. *Soient un motif quasi-additif t et un motif additif linéaire et régulier p. On a :*

1. *Si t est quasi-symbolique, alors $v := t\downarrow_{\mathfrak{R}}$ est soit $\bot$, soit un motif quasi-symbolique.*
2. *$v := t\downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t]\!] = \emptyset$.*
3. *$v := (t \setminus p)\downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t \setminus p]\!] = \emptyset$.*
4. *$v := (t \times p)\downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t \times p]\!] = \emptyset$.*

La preuve est donc donnée par la clause (4) du Lemme précédent. □

En utilisant le système $\mathfrak{R}$, on peut donc enfin vérifier qu'une intersection de sémantique est vide, et ainsi, via la Proposition 3.9, prouver les propriétés d'Exemption de motif d'un terme considéré :

Exemple 4.9. *On a vu dans l'Exemple 4.8, que pour prouver que $u = cons(int(i), flatten(l))$ est exempt de $p_{flat} = cons(lst(l_1), l_2)$, il restait à vérifier que les sémantiques closes des conjonctions $\tilde{u} \times p_{flat}$, $int(i^{-\bot}_{\mathsf{Int}}) \times p_{flat}$, $i^{-\bot}_{\mathsf{Int}} \times p_{flat}$, $(l^{-p_{flat}}_{\mathsf{List}} \setminus \bot) \times p_{flat}$, $(e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)) \times p_{flat}$ et $i^{-p_{flat}}_{\mathsf{Int}} \setminus \bot) \times p_{flat}$ sont vides (avec $\tilde{u} = cons(int(i^{-\bot}_{\mathsf{Int}}), l^{-p_{flat}}_{\mathsf{List}})$). On remarque déjà que les conjonctions $int(i^{-\bot}_{\mathsf{Int}}) \times p_{flat}, i^{-\bot}_{\mathsf{Int}} \times p_{flat}, (e^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)) \times p_{flat}$ et $i^{-p_{flat}}_{\mathsf{Int}} \setminus \bot) \times p_{flat}$ sont mal sortés puisque les motifs à droite et à gauche de la conjonction n'ont pas la même sorte. Ces dernières ont donc bien une sémantique vide.*

Pour les conjonctions restantes, on utilise le système pour vérifier qu'elles peuvent être réduites à $\bot$:

$$\begin{array}{rcl} cons(int(i^{-\bot}_{\mathsf{Int}}), l^{-p_{flat}}_{\mathsf{List}}) \times cons(lst(l_1), l_2) & \xrightarrow{T3} & cons(int(i^{-\bot}_{\mathsf{Int}}) \times lst(l_1), l^{-p_{flat}}_{\mathsf{List}} \times l_2) \\ & \xrightarrow{T4} & cons(\bot, l^{-p_{flat}}_{\mathsf{List}} \times l_2) \\ & \xrightarrow{T2} & cons(\bot, l \mathbin{@} l_2{}^{-p_{flat}}_{\mathsf{List}}) \\ & \xrightarrow{E1} & \bot \end{array}$$

$$
\begin{array}{lll}
(l_{\mathsf{List}}^{-p_{flat}} \setminus \bot) \times cons(lst(l_1), l_2) & \xrightarrow{M2} & l_{\mathsf{List}}^{-p_{flat}} \times cons(lst(l_1), l_2) \\
& \xrightarrow{P1} & l \mathbin{@} (cons(z_1{}_{\mathsf{Expr}}^{-p_{flat}} \times lst(l_1), z_2{}_{\mathsf{List}}^{-p_{flat}} \times l_2) \setminus cons(lst(l_1), l_2)) \\
& \xrightarrow{P1} & l \mathbin{@} (cons(z_1 \mathbin{@} lst(z_3{}_{\mathsf{List}}^{-p_{flat}} \times l_1), z_2{}_{\mathsf{List}}^{-p_{flat}} \times l_2) \setminus cons(lst(l_1), l_2)) \\
& \xrightarrow{T2} & l \mathbin{@} (cons(z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1), z_2 \mathbin{@} l_2) \setminus cons(lst(l_1), l_2)) \\
& \xrightarrow{M7} & l \mathbin{@} (cons((z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1)) \setminus lst(l_1), z_2 \mathbin{@} l_2) \\
& & \quad + cons(z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1), (z_2 \mathbin{@} l_2) \setminus l_2)) \\
& \xrightarrow{M1} & l \mathbin{@} (cons((z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1)) \setminus lst(l_1), z_2 \mathbin{@} l_2) \\
& & \quad + cons(z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1), \bot)) \\
& \xrightarrow{E1} & l \mathbin{@} (cons((z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1)) \setminus lst(l_1), z_2 \mathbin{@} l_2) + \bot) \\
& \xrightarrow{A2} & l \mathbin{@} cons((z_1 \mathbin{@} lst(z_3 \mathbin{@} l_1)) \setminus lst(l_1), z_2 \mathbin{@} l_2) \\
& \xrightarrow{L3} & l \mathbin{@} cons(z_1 \mathbin{@} (lst(z_3 \mathbin{@} l_1) \setminus lst(l_1)), z_2 \mathbin{@} l_2) \\
& \xrightarrow{M7} & l \mathbin{@} cons(z_1 \mathbin{@} lst((z_3 \mathbin{@} l_1) \setminus l_1), z_2 \mathbin{@} l_2) \\
& \xrightarrow{M1} & l \mathbin{@} cons(z_1 \mathbin{@} lst(\bot), z_2 \mathbin{@} l_2) \\
& \xrightarrow{E1} & l \mathbin{@} cons(z_1 \mathbin{@} \bot, z_2 \mathbin{@} l_2) \\
& \xrightarrow{L1} & l \mathbin{@} cons(\bot, z_2 \mathbin{@} l_2) \\
& \xrightarrow{E1} & l \mathbin{@} \bot \xrightarrow{L1} \bot
\end{array}
$$

On peut donc conclure que $\{\!|u|\!\} \cap [\![p_{flat}]\!] = \emptyset$*, et ainsi, d'après Proposition 3.9, que* u *est exempt de* p_{flat}*.*

L'utilisation de l'algorithme `getReachable` et du système $\mathfrak{R}$ définit ainsi une méthode permettant de prouver de façon systématique des propriétés d'Exemption de Motif. L'analyse statique proposée dans ce Chapitre ayant pour but la vérification du CBTRS considéré pour encoder les fonctions de transformation, on souhaite en effet se reposer sur la notion de satisfaction de profil (Définition 3.15). On doit donc montrer que pour chaque règle du CBTRS, pour chaque profil du symbole en tête du membre gauche, toute substitution vérifiant la pré-condition du profil, exprimée en termes d'Exemption de Motif par son membre gauche, vérifie alors sa post-condition, exprimée par son membre droit. Pour un profil considéré, il reste donc encore à proposer une méthode permettant de caractériser les substitutions qui vérifient ainsi la pré-condition.

4.3 Inférence de la forme des variables

Comme on veut fournir une méthode d'analyse statique d'un CBTRS pour prouver que la relation de réécriture induite préserve la sémantique, on se repose sur la notion de satisfaction de profil, qui comme établit dans la Proposition 3.18, admet une relation d'équivalence avec la préservation de sémantique. Cette notion dit qu'une règle satisfait un profil si et seulement respecter la pré-condition décrite par le membre gauche du profil implique de respecter la post-condition décrite par son membre droit. On se concentre donc, dans un premier temps, sur la pré-condition et on cherche à inférer les contraintes qu'elle impose sur les variables de la règle considérée. On pourra ainsi, par la suite, vérifier qu'en prenant en compte ces contraintes le membre droit de la règle vérifie bien la propriété d'Exemption de Motif imposée par le profil.

4.3.1 Substitutions et alias

Pour pouvoir vérifier qu'une règle de réécriture $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$ satisfait un profil $p_1 * \ldots * p_n \mapsto p \in \mathcal{P}$, on propose une méthode permettant de caractériser les substitutions σ telles

que $\sigma(ls_i)$ est exempt de p_i, et de vérifier que $\sigma(rs)$ est exempt de p pour toutes les substitutions correspondantes. On veut, pour cela, inférer les propriétés syntaxiques imposées sur la forme des valeurs substituant les variables de ls_i pour que l'instance correspondante de ls_i soit exempte de p_i. On verra que ces contraintes peuvent en fait être exprimées sous la forme de motifs étendues grâce à un mécanisme se reposant sur l'aliassage des variables dans le membre gauche de la règle considérée.

Dans un motif quasi-symbolique, on veut pouvoir extraire l'information liée aux variables aliassées dans le motif. Pour cela, on définit le motif aliassant chaque variable du motif considéré comme la conjonction des membres droit des alias correspondants.

Définition 4.4. *Étant donné un motif quasi-symbolique t, on note $\mathcal{A}t\mathcal{V}ar(t)$ l'ensemble des variables aliassées de t, défini par $\mathcal{A}t\mathcal{V}ar(t) = \{x \in \mathcal{V}ar(t) \mid \exists \omega \in \mathcal{P}os(t), u \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)\ .\ t_{|\omega} = x \mathbin{@} u\}$.*

De plus, pour toute variable $x \in \mathcal{A}t\mathcal{V}ar(t)$, son motif aliassant est défini par $t_{@x} = \prod_{q \in Q_{@x}} q$ avec $Q_{@x} = \{q \mid \exists \omega \in \mathcal{P}os(t), t_{|\omega} = x \mathbin{@} q\}$.

Étant donné une règle $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$, on va construire à partir de chaque motif ls_i un (ou des) motif(s) analogue(s) telle(s) que toute variable de ls_i est aliassée par un motif quasi-symbolique :

Définition 4.5 (Versions aliassées)**.** *Soit un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on dit qu'un motif τ est une* version aliassée *de t si et seulement si pour toute position $\omega \in \mathcal{P}os(t)$, on a :*

- *si $t(\omega) = f \in \mathcal{F}$, alors $\tau(\omega) = f$;*
- *si $t_{|\omega} = x \in \mathcal{V}ar(t)$, alors $\tau_{|\omega} = x \mathbin{@} q$ avec q un motif quasi-symbolique dont toutes les méta-variables n'apparaissent qu'une seule fois dans τ.*

Pour une telle version aliassées, on note $\sigma_t^{@\tau} = \{x \mapsto \tau_{@x} \mid x \in \mathcal{V}ar(t)\}$.

Exemple 4.10. *On considère l'algèbre de termes définie par la signature Σ_{list} présentée dans l'Exemple 3.5. Pour les motifs $t = cons(e \mathbin{@} (x \setminus lst(l_1)), l \mathbin{@} (cons(int(i), nil) + nil))$ et $u = cons(lst(l@cons(int(i), l_1)), l@cons(e, nil))$, on a $t_{@e} = e@(x \setminus lst(l_1))$, $t_{@l} = l@(cons(int(i), nil) + nil)$ et $u_{@l} = cons(int(i), l_1) \times cons(e, nil)$.*

De plus, t est une version aliassée de $cons(e, l)$, restreignant les variables e, et l, aux termes filtrées par $x \setminus lst(l_2)$ (i.e. des expressions non-listes), et $cons(int(i), nil) + nil)$ (i.e. une liste vide ou contenant un seul entier quelconque), respectivement. Enfin u est une version aliassée de $cons(lst(l), l)$ restreignant la variable l aux termes filtrées par $cons(int(l), l_1)$ (i.e. les listes contenant un entier en tête) et $cons(e, nil)$ (i.e. les listes contenant une seule expression).

On voit dans cet exemple que l'utilisation d'alias permet d'exprimer des contraintes d'instanciation sur les variables du terme considéré. On va donc se servir de cette approche pour encoder l'ensemble des substitutions σ telles que $\sigma(ls_i)$ est exempt du motif p_i du profil considéré, *i.e.* telles que $[\![\sigma(ls_i)]\!] \subseteq [\![x_{s_i}^{-p_i}]\!]$ (Proposition 3.7).

La notion de version aliassée des motifs ls_i permet ainsi de donner une caractérisation des substitutions en inférant les contraintes sur les variables du motif :

Lemme 4.12. *Soit λ une version aliassée d'un terme linéaire $l \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on a : $\forall \sigma\ ([\![\sigma(l)]\!] \subseteq [\![\lambda]\!] \iff \forall x \in \mathcal{V}ar(l), [\![\sigma(x)]\!] \subseteq [\![\lambda_{@x}]\!])$*

Démonstration. On prouve le Lemme par induction sur la forme de l.

Si $l = x_s \in \mathcal{X}$, alors $\lambda = x \mathbin{@} t$ et on a donc $\lambda_{@x} = t$, pour toute substitution σ, $[\![\sigma(l)]\!] \subseteq [\![l_{\mathsf{List}}^{-p_{flat}}]\!]$ si et seulement si $[\![\sigma(x)]\!] \subseteq [\![t]\!]$.

Si $l = c(l_1, \ldots, l_n)$ avec $c \in \mathcal{C}^n$, alors $\lambda = c(\lambda_1, \ldots, \lambda_n)$ avec λ_i une version aliassée de l_i pour tout $i \in [1, n]$. Par induction, on a pour tout $i \in [1, n]$, pour toute substitution σ $[\![\sigma(l_i)]\!] \subseteq [\![\lambda_i]\!]$ si et seulement si $[\![\sigma(x)]\!] \subseteq [\![\lambda_{i@x}]\!]$. On prouve les deux implications séparément :

- On considère une substitution σ telle que $[\![\sigma(l)]\!] \subseteq [\![\lambda]\!]$. D'après Proposition 3.5, on a donc $[\![\sigma(l_i)]\!] \subseteq [\![\lambda_i]\!]$ pour tout $i \in [1, n]$. Donc, par induction, on a, pour tout $i \in [1, n]$, pour toute variable $x \in \mathcal{V}ar(l_i)$, $[\![\sigma(x)]\!] \subseteq [\![\lambda_{i@x}]\!]$. D'où, par linéarité de l, on a, pour toute variable $x \in \mathcal{V}ar(l), [\![\sigma(x)]\!] \subseteq [\![\lambda_{@x}]\!]$.
- On considère une substitution σ telle que $[\![\sigma(l)]\!] \not\subseteq [\![\lambda]\!]$, *i.e.* il existe $v \in [\![\sigma(l)]\!]$ tel que $v \notin [\![\lambda]\!]$. Comme $\sigma(l) = c(\sigma(l_1), \ldots, \sigma(l_n))$ et $v \in [\![\sigma(l)]\!]$, on a d'après Proposition 3.5 $v = c(v_1, \ldots, t_n)$ avec $v_i \in [\![\sigma(l_i)]\!]$ pour tout $i \in [1, n]$. Or $\lambda \not\twoheadrightarrow v$, donc il existe $j \in [1, n]$ tel que $v_j \notin [\![\lambda_j]\!]$. D'où $[\![\sigma(l_i)]\!] \not\subseteq [\![\lambda_i]\!]$ et par hypothèse induction, il existe donc une variable $x \in \mathcal{V}ar(l_i) \subseteq \mathcal{V}ar(l)$ telle que $[\![\sigma(x)]\!] \not\subseteq [\![\lambda_{i@x}]\!]$, et par linéarité $\lambda_{i@x} = \lambda_{@x}$.

□

Pour pouvoir vérifier qu'une règle de réécriture $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$ satisfait un profil $p_1 * \ldots * p_n \mapsto p \in \mathcal{P}$, on va donc chercher à construire des versions aliassées des motifs $ls_1, \ldots, ls_n$ permettant ainsi d'inférer les contraintes sur les variables du membre gauche. Ces contraintes pourront ainsi ensuite être appliquées au membre droit de la règle pour prouver que la post-condition décrite par le membre droit du profil est bien vérifiée.

4.3.2 Règles d'inférence

Pour prouver la préservation de la sémantique du CBTRS considéré par équivalence avec la notion de satisfaction de profil (Proposition 3.18), il faut donc, pour une règle $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$ et chaque profil $p_1 * \ldots * p_n \mapsto p \in \mathcal{P}$, vérifier que pour toute substitution σ telle que $\sigma(ls_i)$ est exempt de p_i, pour tout $i \in [1, n]$, $\sigma(r)$ est exempt de p. Pour un $i \in [1, n]$ donné, on peut alors remarquer que $\sigma(ls_i)$ est exempt de p_i si et seulement $[\![\sigma(ls_i)]\!] \subseteq [\![ls_i]\!] \cap [\![x_{s_i}^{-p_i}]\!] = [\![ls_i \times x_{s_i}^{-p_i}]\!]$ (avec $x_{s_i}^{-p_i}$ une variable fraîche).

En utilisant le Lemme 4.12, on veut donc inférer les contraintes induites par la pré-condition de la satisfaction de profil sur les variables de la règle. Pour cela, on cherche maintenant à exprimer les conjonctions $ls_i \times x_{s_i}^{-p_i}$ comme une somme de versions aliassées $\lambda_i^1 + \cdots + \lambda_i^m$ de ls_i telle que $[\![ls_i \times x_{s_i}^{-p_i}]\!] = [\![\lambda_i^1 + \lambda_i^2 + \cdots + \lambda_i^m]\!]$ et pour toute substitution $\sigma, \sigma(ls_i)$ est exempt de p_i, si et seulement il existe $k \in [1, m]$ tel que $[\![\sigma(ls_i)]\!] \subseteq [\![\lambda_i^k]\!]$. Le Lemme 4.12 garantit alors qu'il existe $k \in [1, m]$ tel que $[\![\sigma(ls_i)]\!] \subseteq [\![\sigma_{ls_i}^{@\lambda_i^k}(ls_i)]\!]$.

Le système $\mathfrak{R}$ présenté dans la Section précédente permet, en effet, d'obtenir une telle décomposition :

Proposition 4.13. *Soient un terme $t \in \mathcal{T}_s(\mathcal{C}, \mathcal{X})$ et un motif additif régulier et linéaire p, $v := (t \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit un motif de la forme $\tau^1 + \tau^2 + \cdots + \tau^m$ avec $\tau^i, i \in [1, m]$ des version aliassées de t. De plus, si t est linéaire, on a :*

- $v = \bot \iff [\![t \times x_s^{-p}]\!] = \emptyset$;
- *si $v = \tau^1 + \tau^2 + \cdots + \tau^m$, alors pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $[\![\sigma(t)]\!] \subseteq [\![x_s^{-p}]\!] \iff \exists k \in [1, m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$.*

Démonstration. Par confluence de $\mathfrak{R}$, on peut étudier les formes normales des différentes combinaisons concernées par la réduction. On prouve ainsi, en Annexe A, le Lemme suivant :

Lemme 4.14. *Soient une version aliassée τ d'un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ de sorte s et un motif additif linéaire et régulier p. On a :*

1. $v := \tau \downarrow_{\mathfrak{R}}$ *est soit* $\bot$*, soit une verion aliassée de* t *;*
2. $v := (\tau \setminus p) \downarrow_{\mathfrak{R}}$ *est soit* $\bot$*, soit une somme de versions aliassées de* t*. De plus, si* t *est linéaire et* $v = \tau^1 + \tau^2 + \cdots + \tau^m$*, alors pour toute substitution* σ *telle que* $\sigma(t) \in \mathcal{T}(\mathcal{F})$*,* $[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] \iff \exists\, k \in [1,m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$ *;*
3. $v := (\tau \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ *est soit* $\bot$*, soit une somme de versions aliassées de* t*. De plus, si* t *est linéaire et* $v = \tau^1 + \tau^2 + \cdots + \tau^m$*, alors pour toute substitution* σ *telle que* $\sigma(t) \in \mathcal{T}(\mathcal{F})$*,* $[\![\sigma(t)]\!] \subseteq [\![x_s^{-p}]\!] \iff \exists\, k \in [1,m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$.

La preuve est donc donnée par la clause (3) du Lemme précédent. □

On obtient ainsi une caractérisation des substitutions considérées, et il nous reste à vérifier que la post-condition de la satisfaction de profil est bien vérifiée dans le contexte de cette caractérisation.

Exemple 4.11. *On considère l'algèbre de termes définie par la signature* Σ_{list}*, et on veut vérifier que le CBTRS* $\mathcal{R}$ *suivant préserve la sémantique :*

$$\begin{cases} \mathit{flatten}(\mathit{nil}) & \rightarrowtriangle & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \rightarrowtriangle & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \rightarrowtriangle & \mathit{concat}(\mathit{flatten}(l), \mathit{flatten}(l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \rightarrowtriangle & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \rightarrowtriangle & l \end{cases}$$

avec les symboles définis annotés $\mathit{flatten}^{!\mathcal{P}_1}$ *et* $\mathit{concat}^{!\mathcal{P}_2}$ *avec* $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ *et* $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$*, où* $p_{flat} = \mathit{cons}(\mathit{lst}(l_1), l_2)$.

Pour les trois premières règles, comme le symbole est annoté par un unique profil dont le membre gauche est $\bot$*, il suffit de vérifier que les membres droits de chaque règle sont exempts de* p_{flat}*. On peut le faire en calculant leur conjonction avec* p_{flat}*, comme cela a été présenté dans la Section précédente.*

Pour vérifier que la règle $\mathit{concat}(\mathit{cons}(e,l),l') \rightarrowtriangle \mathit{cons}(e, \mathit{concat}(l,l'))$ *satisfait le profil* $p_{flat} * p_{flat} \mapsto p_{flat}$*, on utilise la méthode proposée pour obtenir une caractérisation des substitutions considérées par la satisfaction de profil, i.e. telles que* $\sigma(\mathit{cons}(e,l))$ *et* $\sigma(l')$ *sont tous deux exempts de* p_{flat}*.*

Pour toute substitution σ *ainsi considérée, on a* $[\![\sigma(\mathit{cons}(e,l))]\!] \subseteq [\![\mathit{cons}(e,l) \times x_{\mathsf{List}}^{-p_{flat}}]\!]$*. Et avec le système* $\mathfrak{R}$*, on a* $(\mathit{cons}(e,l) \times x_{\mathsf{List}}^{-p_{flat}}) \downarrow_{\mathfrak{R}} = \mathit{cons}(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus \mathit{lst}(l_1)), l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}})$*. D'après le Lemme 4.12,* σ *doit donc vérifier* $[\![\sigma(e)]\!] \subseteq [\![x_{\mathsf{Expr}}^{-p_{flat}} \setminus \mathit{lst}(l_1)]\!]$ *et* $[\![\sigma(l)]\!] \subseteq [\![y_{\mathsf{List}}^{-p_{flat}}]\!]$*, i.e.* σ *ne peut substituer* e *que par des termes exempts de* p_{flat} *et non-filtrés par* $\mathit{lst}(l_1)$*, et* l *par des termes exempts de* p_{flat}*. De plus, on a* $(l' \times z_{\mathsf{List}}^{-p_{flat}}) \downarrow_{\mathfrak{R}} = l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}$*, i.e.* σ *ne peut substituer* l' *que par des termes exempts de* p_{flat}*. Pour vérifier la satisfaction de profil, il reste donc à montrer que pour une telle substitution* σ*, on a* $\sigma(\mathit{cons}(e, \mathit{concat}(l,l')))$ *exempt de* p_{flat}*, ce que l'on peut ici traduire par :* $\{\!\{\mathit{cons}(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus \mathit{lst}(l_1)), \mathit{concat}(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}))\}\!\} \cap [\![p_{flat}]\!] = \emptyset$.

Les substitutions construites à partir des versions aliassées obtenues via le système $\mathfrak{R}$ et le Lemme 4.12, subsituent les variables du membre gauche des règles considérées par des motifs quasi-symboliques. Comme ces dernières définissent la caractérisation recherchée des substitutions considérées par la satisfaction de profil, on souhaite les appliquer au membre droit de règle pour vérifier la post-condition associée. On obtient alors un motif quasi-symbolique contenant des symboles définis (et une version aliassée du membre droit). Pour pouvoir traduire simplement cette post-condition en termes de sémantique, on étend la notion de sémantique close (et par extension de sémantique profonde) pour de tels motifs :

Définition 4.6 (Sémantique close étendue)**.** *Soit $t \in \mathcal{P}(\mathcal{F}, \mathcal{X}^a)$ un motif quasi-symbolique :*

- $[\![t]\!] = \bigcup_\omega \bigcup_v [\![t\,[v]_\omega]\!]$ *avec* $\omega \in \mathcal{P}os(t)$ *telle que* $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ *avec* $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$*, et* $v \in \mathcal{T}_s(\mathcal{C})$ *exempte de* $q = \sum_{q' \in \mathcal{Q}} q'$*,* $\mathcal{Q} = \{p \mid \exists p_1 * \cdots * p_n \mapsto p \in \mathcal{P} \text{ t.q. } \forall i \in [1,n], [\![t_i]\!] \cap [\![p_i]\!] = \emptyset\}$.

Étant donnée la Proposition 3.9, cette définition reste cohérente avec la Définition 3.4 : pour un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ les définitions sont équivalentes.

Grâce à la Proposition 4.10, on peut alors vérifier la satisfaction de profil en calculant, via le système $\mathfrak{R}$, l'intersection de la sémantique profonde de la forme inférée du membre droit de la règle considérée avec la sémantique close du membre droit du profil considéré :

Proposition 4.15. *Soit une règle linéaire* $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$*, et un profil* $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$*. On note* $v := (\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!) \downarrow_{\mathfrak{R}}$*, et on a :*

- *si* $v = \bot$ *alors la règle satisfait le profil* π *;*
- *sinon* $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ *avec chaque* $\lambda_i^j, j \in [1,m]$ *une version aliassée de* $ls_i, i \in [1,n]$*, et la règle satisfait le profil* π *si et seulement si* $\forall k, [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$.

Démonstration. Soit une règle $ls \twoheadrightarrow rs$ avec $ls = f^{!\mathcal{P}}(ls_1, \ldots, ls_n)$ où $f \in \mathcal{D}$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$. On note $v := (\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!) \downarrow_{\mathfrak{R}}$.

Si $v = \emptyset$, alors, d'après la Proposition 4.13, il existe $k \in [1,n]$ tel que $[\![ls_k \times z_k s_k^{-p_k}]\!] = \emptyset$. Donc, pour toute substitution σ, $\sigma(ls_k)$ n'est pas exempt de p_k. Par conséquent, la règle satisfait le profil π.

On considère maintenant que $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ et on montre que la règle satisfait le profil π si et seulement si $\forall k, [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$:

- On suppose que pour tout $k \in [1,m]$ $\forall k, [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$, et on considère une substitution σ telle que $\sigma(ls_i)$ est exempt de p_i, pour tout $i \in [1,n]$. D'après la Proposition 4.13, il existe $k \in [1,m]$ tel que pour tout $i \in [1,n]$, $[\![\sigma(ls_i)]\!] \subseteq [\![\lambda_i^k]\!]$. D'après le Lemme 4.12, on a donc $[\![\sigma(x)]\!] \subseteq [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(x)]\!]$ pour tout $x \in \mathcal{V}ar(ls)$, et de même on a alors $[\![\sigma(rs)]\!] \subseteq [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!]$. Or, comme $[\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$, la Proposition 3.9 garantit que $\sigma(rs)$ est exempt de p.
- On suppose que la règle satisfait le profil. D'après la Proposition 5.4, le système $\mathfrak{R}$ préserve la sémantique, donc pour tout $k \in [1,m]$ et pour tout $i \in [1,n]$, on a $[\![\lambda_i^k]\!] \subseteq [\![x_{s_i}^{-p_i}]\!]$. D'après la Proposition 3.7, $\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(ls_i) = \lambda_i^k$ est exempt de p_i. Par satisfaction de profil, on a donc, pour tout $k \in [1,m]$, $[\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!]$ est exempt de p, d'où, avec la Proposition 3.9, $[\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$.

□

Exemple 4.12. *On continue la vérification du CBTRS* $\mathcal{R}$ *présentée dans l'Exemple 4.11.*

Pour la règle $concat(cons(e,l), l') \twoheadrightarrow cons(e, concat(l, l'))$*, on a vu dans l'Exemple précédent* $\lambda = (\!|cons(e,l) \times x_{\mathsf{List}}^{-p_{flat}}, l' \times z_{\mathsf{List}}^{-p_{flat}}|\!) \downarrow_{\mathfrak{R}} = (\!|cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}), l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!)$*, d'où* $\sigma^{@\lambda}_{(\!|cons(e,l), l'|\!)} = \{e \mapsto e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), l \mapsto l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mapsto l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}\}$*. D'après la Proposition 4.15, pour prouver que la règle satisfait le profil* $p_{flat} * p_{flat} \mapsto p_{flat}$*, il reste*

donc à montrer que $[\![\sigma^{@\lambda}_{\{\!|cons(e,l),l'|\!\}}(cons(e, concat(l, l')))]\!] \cap [\![p_{flat}]\!] = \emptyset$. *Pour cela on peut maintenant utiliser l'algorithme* `getReachable` *et le système* $\mathfrak{R}$ *pour calculer l'équivalent sémantique de* $\sigma^{@\lambda}_{\{\!|cons(e,l),l'|\!\}}(cons(e, concat(l, l'))) = cons(e \mathbin{@} (x^{-p_{flat}}_{\mathsf{Expr}} \setminus lst(l_1)), concat(l \mathbin{@} y^{-p_{flat}}_{\mathsf{List}}, l' \mathbin{@} z^{-p_{flat}}_{\mathsf{List}}))$ *et l'intersection de sémantiques sous-jacente.*

Pour conclure quant à la préservation de sémantique du CBTRS considéré, on étend dans la Section suivante la notion d'équivalent sémantique, introduit avec la Définition 3.13, aux motifs quasi-symboliques de façon similaire à la sémantique close. On aura ainsi proposé une méthode systématique permettant de calculer cet équivalent sémantique et les intersections de sémantique comme présenté dans les Sections précédentes.

4.4 Calcul d'équivalent sémantique

Comme dans la Section 3.2.3, on peut observer que la sémantique d'un terme quasi-symbolique ayant un symbole défini comme symbole de tête dépend uniquement des profils donnés en annotation du symbole et des propriétés d'Exemption de Motif vérifiées par ses sous-termes. Sa sémantique est équivalente à la sémantique d'une variable annotée par le *motif annotant* et on peut généraliser cette approche en construisant, pour tout motif quasi-symbolique de $\mathcal{P}(\mathcal{F}, \mathcal{X}^a)$, un motif quasi-symbolique de $\mathcal{P}(\mathcal{C}, \mathcal{X}^a)$ ayant une sémantique équivalente :

Définition 4.7 (Équivalent sémantique)**.** *Soit* $t \in \mathcal{P}(\mathcal{F}, \mathcal{X}^a)$ *un motif quasi-symbolique, on appelle équivalent sémantique de* t *le motif quasi-symbolique* $\tilde{t} \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$*, construit à partir de* t *:*

- *si* $t = c(t_1, \ldots, t_n)$ *avec* $c \in \mathcal{C}^n$*, alors* $\tilde{t} = c(\tilde{t_1}, \ldots, \tilde{t_n})$ *;*
- *si* $t = \varphi^{!\mathcal{P}}_s(t_1, \ldots, t_n)$ *avec* $\varphi \in \mathcal{D}^n$*, alors* $\tilde{t} = z_s^{-\diamond^{\mathcal{P}}(t_1,\ldots,t_n)}$ *;*
- *sinon* $\tilde{t} = t$.

où $z_s^{-\diamond^{\mathcal{P}}(t_1,\ldots,t_n)}$ *est une variable fraiche et* $\diamond^{\mathcal{P}}(t_1, \ldots, t_n) = \sum_{q \in \mathcal{Q}'}$ *avec* $\mathcal{Q}' = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ *t.q.* $\forall i \in [1, n], [\![\tilde{t_i}]\!] \cap [\![l_i]\!] = \emptyset\}$.

Un motif quasi-symbolique de $\mathcal{P}(\mathcal{F}, \mathcal{X}^a)$ et son équivalent sémantique ont la même sémantique close :

Proposition 4.16. *Soit* $t \in \mathcal{P}(\mathcal{F}, \mathcal{X}^a)$*, on a* $[\![t]\!] = [\![\tilde{t}]\!]$.

Démonstration. La preuve est identique à celle de la Proposition 3.12. □

Comme la sémantique profonde est définie par fermeture de la sémantique close par relation de sous-terme, la propriété est également vraie pour la sémantique profonde

Corollaire 4.17. *Soit* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$*, on a* $\{\!|t|\!\} = \{\!|\tilde{t}|\!\}$.

Démonstration. La preuve immédiate par Proposition 4.16 et par définition de la sémantique profonde. □

Comme on sait décomposer et calculer les sémantiques de motifs constructeurs (cf. Sections 4.1 et 4.2), l'utilisation de l'équivalent sémantique permet ainsi le calcul de la sémantique des termes obtenus par inférence pour la vérification de la satisfaction de profil. De même, pour construire cet équivalent sémantique, on utilise l'algorithme `getReachable` et le système $\mathfrak{R}$, comme présenté dans les Sections 4.1 et 4.2, pour calculer récursivement les intersections de sémantiques profonde et close jusqu'à obtenir l'équivalent sémantique du terme considéré. On peut

ainsi conclure quant à la satisfaction du profil en répétant finalement l'opération sur l'équivalent obtenu.

Exemple 4.13. *On continue la vérification du CBTRS $\mathcal{R}$ présentée dans l'Exemple 4.11.*

Dans l'Exemple 4.12, on a montré que pour vérifier la satisfaction du profil $p_{flat} * p_{flat} \mapsto p_{flat}$ par la règle $concat(cons(e,l),l') \twoheadrightarrow cons(e, concat(l,l'))$, il faut que $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$ avec $\rho = cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), concat(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}))$. Pour vérifier cela, on construit l'équivalent sémantique de $\rho : \tilde{\rho} = cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), \tilde{\tau})$ avec $\tilde{\tau}$ l'équivalent sémantique de $concat^{!\mathcal{P}_2}(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}})$. On doit donc maintenant calculer $\diamond^{\mathcal{P}_2}\left(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}\right)$: comme $l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}$ et $l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}$ ne contiennent pas de symbole défini, ils sont leurs propres équivalents sémantiques, et via l'algorithme `getReachable` *et le système $\mathfrak{R}$ on peut vérifier, comme dans l'Exemple 4.9, que $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \{\!|l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$. Par conséquent, on a $\diamond^{\mathcal{P}_2}\left(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}\right) = p_{flat}$, et donc $\tilde{\rho} = cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), y_{\mathsf{List}}^{-p_{flat}})$.*

Finalement, on peut utiliser l'algorithme `getReachable` *et le système $\mathfrak{R}$, de façon similaire à l'Exemple 4.9 pour vérifier que $[\![\tilde{\rho}]\!] \cap [\![p_{flat}]\!] = \emptyset$. On peut alors conclure que la règle satisfait tous les profils de $\mathcal{P}_2$, et donc que, d'après la Proposition 3.18, elle préserve la sémantique.*

L'utilisation d'équivalents sémantiques permet ainsi de finaliser l'analyse statique de CBTRS pour prouver la préservation de la sémantique par la relation de réécriture induite. On a, en effet, vu que le choix d'un profil induit une propriété de préservation de sémantique qu'il était nécessaire de vérifier. En vérifiant cette dernière, avec la méthode présentée ici, on peut donc valider les profils choisis pour annoter chaque symbole défini. On présente dans la Section suivante une analyse complète du système présenté dans l'Exemple 4.11, pour expliciter clairement la méthode d'analyse statique.

4.5 Méthode d'analyse linéaire

Dans les Sections 4.1 et 4.2, on a présenté comment on peut décomposer la sémantique profonde d'un terme pour en calculer l'intersection avec la sémantique close d'un motif additif linéaire et régulier. On a vu dans la Section 4.3, comment cette technique peut être utilisée pour vérifier la satisfaction de profil d'une règle de réécriture. Le but étant de prouver qu'un CBTRS donné préserve la sémantique, pour vérifier que les profils choisis sont corrects dans le contexte de la relation de réécriture induite par ce dernier.

4.5.1 Présentation de la méthode

Pour prouver qu'un CBTRS donné préserve la sémantique, la Proposition 3.18 établit qu'il est nécessaire et suffisant de montrer que chaque règle satisfait tous les profils du symbole de tête de son membre gauche. On utilise alors la Proposition 4.15 pour vérifier que ces profils sont bien satisfaits.

On peut donc appliquer l'analyse présentée dans les Sections précédentes pour garantir que la sémantique est préservée par la relation de réécriture induite du CBTRS considéré :

Theorem 4.18. *Étant donné un CBTRS linéaire $\mathcal{R}$, si pour toute règle $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs \in \mathcal{R}$ et pour tout profil $p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$, où $p_1, \ldots, p_n, p$ sont des motifs réguliers linéaires*[1]*, en notant $v := \{\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!\}\downarrow_{\mathfrak{R}}$, on a :*

1. On rappelle que, comme présenté dans la Section 2.2.3, on peut toujours mettre un tel motif sous forme additive.

- $v = \bot$;
- *ou* $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ *avec chaque* $\lambda_i^j, j \in [1, m]$ *une version aliassée de* $ls_i, i \in [1, n]$, *et* $\forall k, \{\!|\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)|\!\} \cap [\![p]\!] = \emptyset$;

alors, pour tout termes clos $t, v \in \mathcal{T}(\mathcal{F})$, *on a :*

1. *Si* $t \Longrightarrow^*_{\mathcal{R}} v$, *alors :*
   - 1.1. $[\![v]\!] \subseteq [\![t]\!]$;
   - 1.2. *pour tout motif* q *tel que* t *est exempt de* q, v *est exempt de* q ;
2. *Si* $\mathcal{R}$ *est complet et* $v = t\downarrow_{\mathcal{R}}$, *alors* v *est une valeur et, pour tout motif* q *tel que* t *est exempt de* q, $q \not\lhd v_{|\omega}$ *pour toute position* $\omega \in \mathcal{P}os(v)$.

Démonstration. La preuve de la clause (1.1) est immédiate avec les Propositions 4.15, 3.18 et 3.15.

La clause (1.2) découle directement de la précédente via la Proposition 3.7. Enfin, si $\mathcal{R}$ est complet et v une forme normale, alors v est une valeur, et la clause (2) est donnée par définition de l'Exemption de Motif. □

Pour chaque règle, la méthode d'analyse comporte ainsi trois étapes majeures :

1. une étape d'inférence des substitutions considérées par la notion de satisfaction de profil basée sur la Proposition 4.15 ;
2. une étape de construction de l'équivalent sémantique (Définition 4.7) pour ramener les termes inférés à des motifs constructeurs ;
3. une étape de calcul des sémantiques pour prouver que l'intersection de sémantiques considérée par la Proposition 3.9 est bien vide.

La première étape repose sur le système $\mathfrak{R}$ présenté en Figure 4.8, et la notion d'aliasing décrite par le Lemme 4.12. Étant donné une règle $f_s^{!\mathcal{P}}(ls_1, \cdots, ls_n) \twoheadrightarrow rs$ et un profil $p_1 * \cdots * p_n \mapsto p$, on caractérise les substitutions vérifiant la pré-condition définie par le membre gauche du profil en réduisant avec $\mathfrak{R}$ le n-uplet de conjonctions $(\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!)$. Les Propositions 5.4 et 4.10 garantissent qu'on obtient une somme de n-uplets $(\!|\lambda_1, \ldots, \lambda_n|\!)$ où chaque λ_i est une version aliassée de ls_i, et tel qu'on peut donc extraire la substitution aliassante $\sigma^{@(\!|\lambda_1,\ldots,\lambda_n|\!)}_{(\!|ls_1,\cdots,ls_n|\!)}$. Par aliasing, on doit alors montrer que la post-condition du profil est respectée, en vérifiant que $\{\!|\sigma^{@(\!|\lambda_1,\ldots,\lambda_n|\!)}_{(\!|ls_1,\cdots,ls_n|\!)}(rs)|\!\} \cap [\![p]\!] = \emptyset$.

Pour vérifier cette condition, on doit décomposer la sémantique profonde du terme droit inféré pour ensuite pouvoir utiliser le système $\mathfrak{R}$. La méthode de décomposition présentée dans la Section 4.1, et le système $\mathfrak{R}$ utilisant uniquement des motifs constructeurs, la deuxième étape propose, dans un premier temps de construire un équivalent sémantique du terme considéré. La notion d'équivalent sémantique, originellement introduite dans le Chapitre précédent, est naturellement étendue aux versions aliassées de termes de l'algèbre, et se construit en calculant récursivement les intersections de sémantiques profonde et close de ses sous-termes (comme présenté dans les Sections 4.1 et 4.2).

La troisième étape permet alors enfin de conclure quand à la satisfaction du profil, en répétant les opérations de décomposition de sémantique profonde et de calcul de conjonctions de motifs sur l'équivalent sémantique obtenu.

4.5.2 Cas d'analyse statique d'un système de réécriture

On propose d'étudier en détail le CBTRS $\mathcal{R}$ présenté dans l'Exemple 4.11.

On rappelle que l'on travaille dans l'algèbre de termes définie par la signature $\Sigma_{list} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} \mathsf{Expr} & := & \mathit{int}(\mathsf{Int}) & \mathsf{Int} & := & z & \mathsf{List} & := & \mathit{nil} \\ & | & \mathit{lst}(\mathsf{List}) & & | & s(\mathsf{Int}) & & | & \mathit{cons}(\mathsf{Expr}, \mathsf{List}) \end{array}$$

avec les symboles définis $\mathcal{D} = \{\mathit{flatten}^{!\mathcal{P}_1} : \mathsf{List} \mapsto \mathsf{List}, \mathit{concat}^{!\mathcal{P}_2} : \mathsf{List} * \mathsf{List} \mapsto \mathsf{List}\}$ annotés avec $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ et $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$ où $p_{flat} = \mathit{cons}(\mathit{lst}(l_1), l_2)$.

On veut donc prouver que le CBTRS suivant préserve la sémantique :

$$\left\{ \begin{array}{lcl} \mathit{flatten}(\mathit{nil}) & \twoheadrightarrow & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \twoheadrightarrow & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \twoheadrightarrow & \mathit{concat}(\mathit{flatten}(l), \mathit{flatten}(l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \twoheadrightarrow & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \twoheadrightarrow & l \end{array} \right.$$

Les symboles définis de l'algèbre considérée n'ayant chacun qu'un seul profil, on doit donc montrer pour les trois premières règles qu'elles satisfassent le profil $\bot \mapsto p_{flat}$, et pour les deux dernières le profil $p_{flat} * p_{flat} \mapsto p_{flat}$.

On applique pour chaque règle les trois étapes de la méthode d'analyse :

- règle $\mathit{flatten}(\mathit{nil}) \twoheadrightarrow \mathit{nil}$:

  1. On a
     - $(\mathit{nil} \times x^{-\bot}_{\mathsf{List}}) \downarrow_{\mathfrak{R}} = \mathit{nil}$;
     - $\sigma^{@\mathit{nil}}_{\mathit{nil}} = \{\}$;
     - d'où $\sigma^{@\mathit{nil}}_{\mathit{nil}}(\mathit{nil}) = \mathit{nil}$.

     Donc d'après la Proposition 4.15, la règle satisfait le profil $\bot \mapsto p_{flat}$ si et seulement si $\{\!|\mathit{nil}|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
  2. On a $\tilde{\mathit{nil}} = \mathit{nil}$.
  3. D'après la Proposition 4.1, on a $\{\!|\mathit{nil}|\!\} = [\![\mathit{nil}]\!]$, d'où $\{\!|\mathit{nil}|\!\} \cap [\![p_{flat}]\!] = [\![\mathit{nil} \times p_{flat}]\!]$. Enfin, on a $(\mathit{nil} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|\mathit{nil}|\!\} \cap [\![p_{flat}]\!] = \emptyset$, donc la règle satisfait bien le profil $\bot \mapsto p_{flat}$.

- règle $\mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) \twoheadrightarrow \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l))$:

  1. On a
     - $\lambda = (\mathit{cons}(\mathit{int}(n), l) \times x^{-\bot}_{\mathsf{List}}) \downarrow_{\mathfrak{R}} \mathit{cons}(\mathit{int}(n \mathbin{@} y^{-\bot}_{\mathsf{Int}}), l \mathbin{@} z^{-\bot}_{\mathsf{List}})$;
     - $\sigma^{@\lambda}_{\mathit{cons}(\mathit{int}(n),l))} = \{n \mapsto n \mathbin{@} y^{-\bot}_{\mathsf{Int}}, l \mapsto l \mathbin{@} z^{-\bot}_{\mathsf{List}}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{\mathit{cons}(\mathit{int}(n),l))}(\mathit{cons}(\mathit{int}(n), \mathit{flatten}(l))) = \mathit{cons}(\mathit{int}(n \mathbin{@} y^{-\bot}_{\mathsf{Int}}), \mathit{flatten}(l \mathbin{@} z^{-\bot}_{\mathsf{List}}))$.

     Donc d'après la Proposition 4.15, la règle satisfait le profil $\bot \mapsto p_{flat}$ si et seulement si $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
  2. On calcule donc l'équivalent sémantique de ρ :

- Étant donné que le seul profil du symbole *flatten* est $\bot \mapsto p_{flat}$, on veut vérifier si $\{\!|l \mathbin{@} z_{\mathsf{List}}^{-\bot}|\!\} \cap [\![\bot]\!] = \emptyset$, ce qui est trivial puisque $[\![\bot]\!] = \emptyset$.
- On a donc $\tilde{\rho} = cons(int(n \mathbin{@} y_{\mathsf{Int}}^{-\bot}), l_{\mathsf{List}}^{-p_{flat}})$.

3. D'après la Proposition 4.1, on a $\{\!|\tilde{\rho}|\!\} = [\![\tilde{\rho}]\!] \cup [\![int(n \mathbin{@} y_{\mathsf{Int}}^{-\bot})]\!] \cup \{\!|y_{\mathsf{Int}}^{-\bot}|\!\} \cup \{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\}$, et, avec l'algorithme `getReachable`, $\{\!|y_{\mathsf{Int}}^{-\bot}|\!\} = [\![y_{\mathsf{Int}}^{-\bot}]\!]$ et $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$. On a donc, en éliminant les intersections de sortes différentes :

$$\begin{aligned}
\{\!|\rho|\!\} \cap [\![p_{flat}]\!] &= ([\![\tilde{\rho}]\!] \cup [\![int(n \mathbin{@} y_{\mathsf{Int}}^{-\bot}]\!] \cup [\![y_{\mathsf{Int}}^{-\bot}]\!] \cup [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!]) \cap [\![p_{flat}]\!] \\
&= ([\![\tilde{\rho}]\!] \cap [\![p_{flat}]\!]) \cup ([\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cap [\![p_{flat}]\!]) \\
&\overset{Pr\,3.8}{=} [\![\tilde{\rho} \times p_{flat}]\!] \cup [\![l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]
\end{aligned}$$

De plus, on peut vérifier que :

- $(\tilde{\rho} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$;
- $(l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$.

Par conséquent, $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$, donc la règle satisfait bien le profil $\bot \mapsto p_{flat}$.

- règle $flatten(cons(lst(l), l')) \twoheadrightarrow concat(flatten(l), flatten(l'))$:

1. On a
   - $\lambda = (cons(lst(l), l') \times x_{\mathsf{List}}^{-\bot}) \downarrow_{\mathfrak{R}} = cons(lst(l \mathbin{@} y_{\mathsf{List}}^{-\bot}), l' \mathbin{@} z_{\mathsf{List}}^{-\bot})$;
   - $\sigma^{@\lambda}_{cons(lst(l),l')} = \{l \mapsto l \mathbin{@} y_{\mathsf{List}}^{-\bot}, l' \mapsto l' \mathbin{@} z_{\mathsf{List}}^{-\bot}\}$;
   - d'où $\rho = \sigma^{@\lambda}_{cons(lst(l),l')}(concat(flatten(l), flatten(l'))) = concat(flatten(l \mathbin{@} y_{\mathsf{List}}^{-\bot}), flatten(l' \mathbin{@} z_{\mathsf{List}}^{-\bot}))$.

   Donc d'après la Proposition 4.15, la règle satisfait le profil $\bot \mapsto p_{flat}$ si et seulement si $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
2. On calcule donc l'équivalent sémantique de ρ :
   - Étant donné que le seul profil du symbole *concat* est $p_{flat} * p_{flat} \mapsto p_{flat}$, on veut calculer les équivalents sémantiques de $r_1 = flatten(l \mathbin{@} y_{\mathsf{List}}^{-\bot})$ et $r_2 = flatten(l' \mathbin{@} z_{\mathsf{List}}^{-\bot})$.
   - Étant donné que le seul profil du symbole *flatten* est $\bot \mapsto p_{flat}$, on veut vérifier si $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-\bot}|\!\} \cap [\![\bot]\!] = \emptyset$ et $\{\!|l' \mathbin{@} z_{\mathsf{List}}^{-\bot}|\!\} \cap [\![\bot]\!] = \emptyset$, ce qui est trivial puisque $[\![\bot]\!] = \emptyset$.
   - On a donc $\tilde{r_1} = \tilde{r_2} = l_{\mathsf{List}}^{-p_{flat}}$, et on veut vérifier si $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$. Avec l'algorithme `getReachable`, on a $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$, d'où $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = [\![l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]$, en éliminant les conjonctions mal-sortées. Enfin, on peut vérifier que $(l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
   - On a donc $\tilde{\rho} = l_{\mathsf{List}}^{-p_{flat}}$.
3. Avec l'algorithme `getReachable`, on a $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$ d'où $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = [\![l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]$, en éliminant les conjonctions mal-sortées. Enfin, on peut vérifier que $(l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$. Donc la règle satisfait bien le profil $\bot \mapsto p_{flat}$.

- règle $concat(cons(e,l),l') \twoheadrightarrow cons(e,concat(l,l'))$:

  1. On a
     - $\lambda = (\!|cons(e,l) \times x_{\mathsf{List}}^{-p_{flat}}, l' \times z_{\mathsf{List}}^{-p_{flat}}|\!) \downarrow_{\mathfrak{R}}$
       $= (\!|cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}), l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!)$;
     - $\sigma^{@\lambda}_{(\!|cons(e,l),l'|\!)} = \{e \mapsto e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), l \mapsto l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mapsto l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{(\!|cons(e,l),l'|\!)}(cons(e,concat(l,l')))$
       $= cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), concat(l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}, l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}))$.

     Donc d'après la Proposition 4.15, la règle satisfait le profil $p_{flat} * p_{flat} \mapsto p_{flat}$ si et seulement si $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
  2. On calcule donc l'équivalent sémantique de ρ :
     - Étant donné que le seul profil du symbole *concat* est $p_{flat} * p_{flat} \mapsto p_{flat}$, on veut vérifier si $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$ et $\{\!|l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
     - Avec l'algorithme `getReachable`, on a $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}|\!\} = \{\!|l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$, d'où $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \{\!|l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = [\![l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]$, en éliminant les conjonctions mal-sortées. Enfin, on peut vérifier que $(l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|l \mathbin{@} y_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \{\!|l' \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
     - On a donc $\tilde{\rho} = cons(e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)), l_{\mathsf{List}}^{-p_{flat}})$.
  3. D'après la Proposition 4.1, on a $\{\!|\tilde{\rho}|\!\} = [\![\tilde{\rho}]\!] \cup [\![int(n \mathbin{@} y_{\mathsf{Int}}^{-\bot})]\!] \cup \{\!|e \mathbin{@} (x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1))|\!\} \cup \{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\}$, et, avec l'algorithme `getReachable`, $\{\!|x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)|\!\} = [\![x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup \{\!|i_{\mathsf{Int}}^{-p_{flat}}|\!\} \cup [\![l_{\mathsf{List}}^{-p_{flat}}]\!]$ et $\{\!|l_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![l_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![x_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$. On a donc $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = [\![\tilde{\rho} \times p_{flat}]\!] \cup [\![l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]$, en éliminant les conjonctions mal-sortées. De plus, on peut vérifier que :
     - $(\tilde{\rho} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$;
     - $(l_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$.

     Par conséquent, $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$, donc la règle satisfait bien le profil $p_{flat} * p_{flat} \mapsto p_{flat}$.

- règle $concat(nil,l) \twoheadrightarrow l$:

  1. On a
     - $\lambda = (\!|nil \times x_{\mathsf{List}}^{-p_{flat}}, l \times z_{\mathsf{List}}^{-p_{flat}}|\!) \downarrow_{\mathfrak{R}} = (\!|nil, l \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}|\!)$;
     - $\sigma^{@\lambda}_{(\!|nil,l|\!)} = \{l \mapsto l \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{(\!|nil,l|\!)}(l) = l \mathbin{@} z_{\mathsf{List}}^{-p_{flat}}$.

     Donc d'après la Proposition 4.15, la règle satisfait le profil $p_{flat} * p_{flat} \mapsto p_{flat}$ si et seulement si $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$.
  2. On a $\tilde{\rho} = \rho$.
  3. Avec l'algorithme `getReachable`, on a $\{\!|z_{\mathsf{List}}^{-p_{flat}}|\!\} = [\![z_{\mathsf{List}}^{-p_{flat}}]\!] \cup [\![e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)]\!] \cup [\![i_{\mathsf{Int}}^{-p_{flat}}]\!]$. On a donc $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = [\![z_{\mathsf{List}}^{-p_{flat}} \times p_{flat}]\!]$, en éliminant les conjonctions mal-sortées.

Enfin, on peut vérifier que $(z_{\mathsf{List}}^{-p_{flat}} \times p_{flat}) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|\rho|\!\} \cap [\![p_{flat}]\!] = \emptyset$. Donc la règle satisfait bien le profil $p_{flat} * p_{flat} \mapsto p_{flat}$.

On en déduit donc que le système $\mathcal{R}$ préserve la sémantique.

Un cas qui n'est pas présenté dans l'analyse précédente est le cas où une règle n'est pas compatible avec un profil du symbole de tête : quand il n'existe pas de substitution tel que les sous-termes à gauche vérifient le profil. On peut observer un tel cas en ajoutant le profil $cons(e,l) * cons(e,l) \mapsto cons(e,l)$ dans l'annotation $\mathcal{P}_2$ du symbole *concat* (*i.e.* la concaténation de deux listes vides est une liste vide). Il faut alors vérifier que les deux dernières règles vérifient également ce profil [1] :

- règle $concat(cons(e,l),l') \rightarrowtriangle cons(e,concat(l,l'))$:
  On a $(\!|cons(e,l) \times x_{\mathsf{List}}^{-cons(e,l)}, l' \times z_{\mathsf{List}}^{-cons(e,l)}|\!) \downarrow_{\mathfrak{R}} = \bot$ Donc d'après la Proposition 4.15, la règle satisfait le profil $cons(e,l) * cons(e,l) \mapsto cons(e,l)$.

- règle $concat(nil,l) \rightarrowtriangle l$:

  1. On a
     - $\lambda = (\!|nil \times x_{\mathsf{List}}^{-cons(e,l)}, l \times z_{\mathsf{List}}^{-cons(e,l)}|\!) \downarrow_{\mathfrak{R}} = (\!|nil, l \mathbin{@} z_{\mathsf{List}}^{-cons(e,l)}|\!)$;
     - $\sigma^{@\lambda}_{(\!|nil,l|\!)} = \{l \mapsto l \mathbin{@} z_{\mathsf{List}}^{-cons(e,l)}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{(\!|nil,l|\!)}(l) = l \mathbin{@} z_{\mathsf{List}}^{-cons(e,l)}$.

     Donc d'après la Proposition 4.15, la règle satisfait le profil $cons(e,l) * cons(e,l) \mapsto cons(e,l)$ si et seulement si $\{\!|\rho|\!\} \cap [\![cons(e,l)]\!] = \emptyset$.
  2. On a $\tilde{\rho} = \rho$.
  3. Avec l'algorithme `getReachable`, on a $\{\!|z_{\mathsf{List}}^{-cons(e,l)}|\!\} = [\![z_{\mathsf{List}}^{-cons(e,l)}]\!]$. On a donc $\{\!|\rho|\!\} \cap [\![cons(e,l)]\!] = [\![z_{\mathsf{List}}^{-cons(e,l)} \times cons(e,l)]\!]$, en éliminant les conjonctions mal-sortées. Enfin, on peut vérifier que $(z_{\mathsf{List}}^{-cons(e,l)} \times cons(e,l)) \downarrow_{\mathfrak{R}} = \bot$, d'où $\{\!|\rho|\!\} \cap [\![cons(e,l)]\!] = \emptyset$. Donc la règle satisfait bien le profil $cons(e,l) * cons(e,l) \mapsto cons(e,l)$.

On peut ici bien observer que la règle $concat(cons(e,l),l') \rightarrowtriangle cons(e,concat(l,l'))$ ne s'applique jamais dans le cas où les sous-termes sont exempts de $cons(e,l)$, puisque pour toute substitution σ, le premier sous-terme $\sigma(cons(e,l))$ ne peut pas être exempt de $cons(e,l)$. La règle satisfait donc bien le profil puisque ce dernier ne s'appliquera jamais.

On voit de plus que la méthode d'analyse est assez répétitive et facilement automatisable. Une implémentation de cette méthode sera discutée en détail dans le Chapitre 6.

4.6 Synthèse

Dans ce Chapitre nous avons présenté une méthode d'analyse statique permettant de vérifier que les annotations choisies pour décorer les symboles définis sont en accord avec le CBTRS considéré : il ne faut pas que la relation de réécriture induite par le CBTRS admette une réduction contredisant le comportement décrit par l'un des profils choisis. Cette méthode repose

1. les calculs d'équivalent sémantique sont également légèrement différents puisqu'il faut prendre en compte les deux profils de *concat*

sur l'équivalence entre la préservation de la sémantique (Définition 3.14) et de la satisfaction de profil (Définition 3.15), établie par la Proposition 3.18. La méthode cherche donc, pour chaque règle du CBTRS, à vérifier qu'elle satisfait tous les profils de l'annotation du symbole de tête de son membre gauche.

Pour chaque règle du CBTRS, et chaque profil considéré, la méthode peut être décrite en trois étapes majeurs :

1. une étape d'inférence des substitutions satisfaisant les pré-conditions du profil, basée sur la Proposition 4.15 ;
2. une étape de construction de l'équivalent sémantique sur-approximant les instances correspondantes du membre droit de la règle ;
3. une étape de vérification pour prouver que la sur-approximation ainsi exprimée satisfait la post-condition du profil.

Pour permettre l'implémentation de ces différentes étapes, un certain nombre de mécanismes ont du être mis en place.

Premièrement, pour étudier les propriétés d'Exemption de Motif et la sémantique profonde des termes, on a introduit une méthode reposant sur une représentation des sémantiques sous la forme de graphes. À partir de cette représentation, on a ainsi pu proposer l'algorithme `getReachable` (Figure 4.7) permettant non seulement de vérifier qu'une sémantique profonde est non-vide, mais également d'en donner une décomposition sous la forme d'une union de sémantiques closes.

Pour prouver la satisfaction de profil, on a introduit un mécanisme basé sur l'aliasing des variables permettant d'inférer les instances du membre gauche de la règle satisfaisant les pré-conditions du profil. Pour cela, on s'est inspiré de l'approche présentée dans [CM19] pour proposer un ensemble de règles de réécriture des motifs étendus, préservant leur sémantique. Le système $\mathfrak{R}$ (Figure 4.8) obtenu permet non seulement d'expliciter les contraintes d'instanciation nécessaires à l'inférence, mais également de vérifier, après construction de l'équivalent sémantique, que les instances correspondantes du membre droit de la règle considérée respectent la post-condition du profil.

La méthode d'analyse ainsi obtenue a été prouvée correcte sous hypothèse de linéarité des termes considérés. L'approche sémantique est néanmoins capable, grâce au mécanisme d'aliasing, de prendre en compte les contraintes liées à la non-linéarité des termes. On propose donc, dans le Chapitre suivant, de présenter comment la méthode présentée ici s'applique aux systèmes non-linéaires, et avec quelles limitations. En réponse à ces dernières, on proposera alors une adaptation du formalisme de sémantique aux termes non-linéaires. On verra ainsi comment les mécanismes présentés dans ce Chapitre s'appliquent à ce nouveau formalisme, et on en déduira une méthode d'analyse pour la vérification de systèmes non-linéaires.

Fonction `getTotalReach`(s, p, r, V, E)
- **Entrées :** s : sorte,
  p : motif annotant,
  r : motif complément,
  V : ensemble de nœuds variables atteints (initialisé à $\emptyset$),
  E : ensemble d'arêtes multiples connues (initialisé à $\emptyset$)
- **Résultat :** couple (V, E) graphe total de $x_s^{-p} \setminus r$
- $r \longleftarrow r + p$
- **si** $\exists\,(s, r') \in V.[\![r']\!] = [\![r]\!]$ **alors retourner** (V, E)
- $V \longleftarrow V \cup \{(s, r)\}$
- **pour** $c \in \mathcal{C}_s$ **faire**
  - $Q_c \longleftarrow$ `computeQc`$(c,\ r)$
  - $(s_1, \ldots, s_n) \longleftarrow \mathcal{D}om(c)$
  - **pour** $(q_1, \ldots, q_n) \in Q_c$ **faire**
    - $E \longleftarrow E \cup \{((s, r), ((s_1, q_1), \ldots, (s_n, q_n)))\}$
    - **pour** $i = 1$ **à** n **faire**
      - $(V, E) \longleftarrow$ `getTotalReach`(s_i, p, q_i, V, E)
- **retourner** (V, E)

Fonction `computeReach`(s, r, V, E, R)
- **Entrées :** u : nœud variable (couple (s, r)),
  V : ensemble de nœuds instanciables,
  E : ensemble d'arêtes multiples,
  R : ensemble de nœuds atteints (initialisé à $\emptyset$)
- **Résultat :** ensemble de nœuds atteignables depuis u
- **si** $(s, r) \in R$ **alors retourner** R
- $R \longleftarrow R \cup \{(s, r)\}$
- **pour** $(u, (v_1, \ldots, v_n)) \in E$ **faire**
  - **si** $\forall i \in [1, n].v_i \in V$ **alors**
    - **pour** $i = 1$ **à** n **faire** $R \longleftarrow$ `computeReach`(v_i, V, E, R)
- **retourner** R

Fonction `getReachable`(s, p, r)
- **Entrées :** s : sorte,
  p : motif annotant,
  r : motif complément
- **Résultat :** ensemble de nœuds variables instanciables et atteignables depuis $x_s^{-p} \setminus r$
- $A \longleftarrow \emptyset$
- $(V, E) \longleftarrow$ `getTotalReach`$(s, p, r, \emptyset, \emptyset)$
- $stable \longleftarrow True$
- **pour** $(s', r') \in V$ **faire**
  - **si** $\exists\, c \in \mathcal{C}_{s'}.(arity\,(c) = 0 \wedge$ `computeQc`$(c, r+p) \neq \emptyset)$ **alors**
    - $A \longleftarrow A \cup \{(s', r')\}$ $stable \longleftarrow False$
- **tant que** $\neg stable$ **faire**
  - $stable \longleftarrow True$
  - **pour** $((s', r'), ((s_1, q_1), \ldots, (s_n, q_n))) \in E$ **faire**
    - **si** $(s', r') \notin A \wedge (\forall i \in [1, n].(s_i, q_i) \in A)$ **alors**
      - $A \longleftarrow A \cup \{(s', r')\}$
      - $stable \longleftarrow False$
- **si** $(s, r) \notin A$ **alors retourner** $\emptyset$
- **retourner** `computeReach`$(s, r, A, E, \emptyset)$

FIGURE 4.7 – Algorithme `getReachable`, et fonctions auxiliaires

Élimine ensemble vide :

(A1) $\bot + \overline{v} \Rightarrow \overline{v}$

(A2) $\overline{v} + \bot \Rightarrow \overline{v}$

(E1) $\delta(\overline{v_1}, \ldots, \bot, \ldots, \overline{v_n}) \Rightarrow \bot$

(E2) $\bot \times \overline{v} \Rightarrow \bot$

(E3) $\overline{v} \times \bot \Rightarrow \bot$

Développe additions :

(S1) $\delta(\overline{v_1}, \ldots, \overline{v_i} + \overline{w_i}, \ldots, \overline{v_n}) \Rightarrow \delta(\overline{v_1}, \ldots, \overline{v_i}, \ldots, \overline{v_n}) + \delta(\overline{v_1}, \ldots, \overline{w_i}, \ldots, \overline{v_n})$

(S2) $(\overline{v_1} + \overline{v_2}) \times \overline{w} \Rightarrow (\overline{v_1} \times \overline{w}) + (\overline{v_2} \times \overline{w})$

(S3) $\overline{v} \times (\overline{w_1} + \overline{w_2}) \Rightarrow (\overline{v} \times \overline{w_1}) + (\overline{v} \times \overline{w_2})$

(S4) $\overline{u} + (\overline{v} + \overline{w}) \Rightarrow (\overline{u} + \overline{v}) + \overline{w}$

Simplifie compléments :

(M1) $\overline{v} \setminus \overline{x}_s^{-\bot} \Rightarrow \bot$

(M2) $\overline{v} \setminus \bot \Rightarrow \overline{v}$

(M3') $(\overline{v_1} + \overline{v_2}) \setminus \overline{w} \Rightarrow (\overline{v_1} \setminus \overline{w}) + (\overline{v_2} \setminus \overline{w})$

(M5) $\bot \setminus \overline{v} \Rightarrow \bot$

(M6') $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus (\overline{v} + \overline{w}) \Rightarrow (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \overline{v}) \setminus \overline{w}$

(M7) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n}) \Rightarrow \alpha(\overline{v_1} \setminus \overline{t_1}, \ldots, \overline{v_n}) + \cdots + \alpha(\overline{v_1}, \ldots, \overline{v_n} \setminus \overline{t_n})$

(M8) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \beta(\overline{w_1}, \ldots, \overline{w_m}) \Rightarrow \alpha(\overline{v_1}, \ldots, \overline{v_n})$ *avec* $\alpha \neq \beta$

Simplifie conjonctions :

(T1) $\overline{x}_s^{-\bot} \times \overline{y}_s^{-\overline{q}} \Rightarrow \overline{x} \mathbin{@} \overline{y}_s^{-\overline{q}}$

(T2) $\overline{x}_s^{-\overline{p}} \times \overline{y}_s^{-\overline{q}} \Rightarrow \overline{x} \mathbin{@} \overline{y}_s^{-\overline{p}}$ *avec* $\overline{p} = \overline{q} \vee \overline{q} = \bot$

(T3) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \alpha(\overline{w_1}, \ldots, \overline{w_n}) \Rightarrow \alpha(\overline{v_1} \times \overline{w_1}, \ldots, \overline{v_n} \times \overline{w_n})$

(T4) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \beta(\overline{w_1}, \ldots, \overline{w_m}) \Rightarrow \bot$ *avec* $\alpha \neq \beta$

Simplifie alias :

(L1) $\overline{x} \mathbin{@} \bot \Rightarrow \bot$

(L2) $\overline{x} \mathbin{@} (\overline{v} + \overline{w}) \Rightarrow \overline{x} \mathbin{@} \overline{v} + \overline{x} \mathbin{@} \overline{w}$

(L3) $(\overline{x} \mathbin{@} \overline{v}) \setminus \overline{w} \Rightarrow \overline{x} \mathbin{@} (\overline{v} \setminus \overline{w})$

(L4) $(\overline{x} \mathbin{@} \overline{v}) \times \overline{w} \Rightarrow \overline{x} \mathbin{@} (\overline{v} \times \overline{w})$

(L5) $\overline{v} \times (\overline{x} \mathbin{@} \overline{w}) \Rightarrow \overline{x} \mathbin{@} (\overline{v} \times \overline{w})$ *avec* $\overline{v} \neq \overline{y} \mathbin{@} \overline{u}$

Controle annotation :

(P1) $\overline{x}_s^{-\overline{p}} \times \alpha(\overline{v_1}, \ldots, \overline{v_n}) \Rightarrow \overline{x} \mathbin{@} (\alpha(z_{1s_1}^{-\overline{p}} \times \overline{v_1}, \ldots, z_{ns_n}^{-\overline{p}} \times \overline{v_n}) \setminus \overline{p})$

(P2) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \overline{x}_s^{-\overline{p}} \Rightarrow \alpha(\overline{v_1} \times z_{1s_1}^{-\overline{p}}, \ldots, \overline{v_n} \times z_{ns_n}^{-\overline{p}}) \setminus \overline{p}$

(P3) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times (\overline{x}_s^{-\overline{p}} \setminus \overline{t}) \Rightarrow (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \overline{x}_s^{-\overline{p}}) \setminus \overline{t}$ si $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} \neq \emptyset$

(P4) $\overline{y}_s^{-\overline{q}} \times (\overline{x}_s^{-\overline{p}} \setminus \overline{t}) \Rightarrow (\overline{y}_s^{-\overline{q}} \times \overline{x}_s^{-\overline{p}}) \setminus \overline{t}$ si $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} \neq \emptyset$

(P5) $(\overline{x}_s^{-\overline{p}} \setminus \overline{t}) \times \overline{v} \Rightarrow (\overline{x}_s^{-\overline{p}} \times \overline{v}) \setminus \overline{t}$ si $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} \neq \emptyset$

(P6) $(\overline{x}_s^{-\overline{p}} \setminus \overline{t}) \setminus \overline{q} \Rightarrow \overline{x}_s^{-\overline{p}} \setminus (\overline{t} + \overline{q})$ si $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} \neq \emptyset$

(P7) $\overline{x}_s^{-\overline{p}} \setminus \overline{t} \Rightarrow \bot$ si $\{\!\{\overline{x}_s^{-\overline{p}} \setminus \overline{t}\}\!\} = \emptyset$

FIGURE 4.8 – $\mathfrak{R}$: réduction de motifs de la forme $u \times v$; $\overline{u}, \overline{v}, \overline{v_1}, \ldots, \overline{v_n}$, $\overline{w}, \overline{w_1}, \ldots, \overline{w_n}$ filtre les motifs quasi-additifs, $\overline{p}, \overline{q}, \overline{t}$ filtre les motifs reguliers additifs, $\overline{t_1}, \ldots, \overline{t_n}$ filtre les motifs symbolics, $\overline{x}, \overline{y}$ filtre les variables. α, β s'étend à tous les symboles de $\mathcal{C}$, et δ de $\mathcal{C}^{n>0}$.

5

Analyse de systèmes non-linéaires

La méthode d'analyse statique présentée dans le Chapitre précédent permet de vérifier que les annotations choisies pour décorer les symboles définis sont en accord avec le CBTRS considéré. Cette méthode se base sur la notion de sémantique introduite dans le Chapitre 3 pour garantir les propriétés d'Exemption de Motif implicitement requises par les profils donnés en annotation. En effet, on a montré que vérifier la préservation de la sémantique par la relation de réécriture induite par le système étudié permet de démontrer la préservation des propriétés d'Exemption de Motif, donc dans le cas d'un terme ayant un symbole défini comme symbole de tête, que toutes ses réductions possibles préserveront les propriétés d'Exemption de Motif garanties par l'annotation de ce symbole.

En pratique, les propriétés de préservation de sémantique (Définition 3.14) et de satisfaction de profil (Définition 3.15), sur lesquelles est basée la méthode d'analyse, sont dépendantes de la préservation de la sémantique et des propriétés d'Exemption de Motif par substitution. Cependant, comme on l'a vu dans le Chapitre 3, cette préservation n'est pas vérifiée pour les termes non-linéaires. La méthode d'inférence proposée dans le Chapitre précédent préservant la sémantique des membres droits des règles de réécriture, elle ne permet donc pas d'inférer correctement le comportement des substitutions considérées dans le cas de membres droits non-linéaires.

En première approximation, deux approches permettent tout de même l'application de la méthode à l'étude de systèmes non-linéaires :

- linéariser les membres droits des règles du système pour obtenir une sur-approximation plus large mais correcte des instances considérées ;
- se restreindre à l'étude de substitutions valeurs, et donc à une relation de réécriture stricte, puisqu'on a vu que celles-ci préservaient toujours la sémantique et les propriétés d'Exemption de Motif (Proposition 3.19).

Cependant, l'intérêt de l'approche sémantique utilisée est qu'elle permet de considérer de façon précise les formes normales potentielles des termes étudiés, en prenant notamment en compte les contraintes liées à la corrélation entre deux instances d'une même variable dans un terme non-linéaire.

On étudiera donc dans un premier temps ces deux approches et leurs limitations. En réponse à ces limitations, on proposera dans la Section 5.2 une adaptation du formalisme présentée dans le Chapitre 3 pour l'analyse des systèmes non-linéaires. On explicitera finalement dans la Section 5.4 comment la méthode d'analyse s'applique dans le contexte de ce formalisme adapté. Finalement, on fera un récapitulatif de la méthode d'analyse non-linéaire, que l'on illustrera sur un cas d'étude.

5.1 Étude de systèmes non-linéaires et limitations

Dans le Chapitre précédent, on a présenté une méthode d'analyse d'un système de réécriture se reposant l'étude de la sémantique de termes linéaires. Dans le cas de termes non-linéaires, il faut prendre en compte les corrélations entre différentes instances d'une même variable dans le motif considéré qui sont ignorées dans cette méthode. Il n'est donc pas possible d'appliquer directement la méthode sur un système non-linéaire. On propose néanmoins deux approches permettant d'utiliser la méthode pour l'analyse de systèmes non-linéaires :

- une approche d'analyse par linéarisation qui permet d'étudier un système non-linéaire en sur-approximant la sémantique de ses membres droits en les linéarisant ;
- une approche d'analyse stricte qui utilise la technique d'aliasing, proposée pour l'inférence de contraintes sur les variables, pour représenter et calculer l'impact des contraintes de corrélation des variables dans un terme constructeur.

5.1.1 Analyse par linéarisation

La méthode d'analyse présentée dans le Chapitre précédent étant conçue pour l'analyse d'un CBTRS linéaire, une approche naturelle permettant de l'utiliser dans le cas où le système considéré est non-linéaire est de ramener l'analyse de ce dernier à l'étude d'un système linéaire.

Pour cela, on peut remarquer pour tout motif étendu $p \in \mathcal{P}(\mathcal{F}, \mathcal{X}^a)$, on peut construire une forme linéarisée de p en remplaçant toutes les méta-variables de p (cf. Définition 1.14) par des variables fraîches ayant la même annotation et la même sorte :

Définition 5.1 (Forme linéarisée)**.** *Étant donné un motif étendu $p \in \mathcal{P}(\mathcal{F}, \mathcal{X}^a)$, on définit la forme linéarisée de p, notée $\mathcal{L}(p)$, comme le motif obtenu en remplaçant toutes les méta-variables de p par des variables fraîches ayant la même annotation et la même sorte.*

On peut ainsi construire $\mathcal{L}(p)$ récursivement :

- *si $p = x_s^{-p}$, alors $\mathcal{L}(p) = z_s^{-p}$ avec z_s^{-p} une variable fraîche ;*
- *si $p = f(p_1, \ldots, p_n)$ avec $f \in \mathcal{F}^n$, alors $\mathcal{L}(p) = f(\mathcal{L}(p_1), \ldots, \mathcal{L}(p_n))$;*
- *si $p = p_1 + p_2$, alors $\mathcal{L}(p) = \mathcal{L}(p_1) + \mathcal{L}(p_2)$;*
- *si $p = p_1 \setminus p_2$, alors $\mathcal{L}(p) = \mathcal{L}(p_1) \setminus \mathcal{L}(p_2)$;*
- *si $p = p_1 \times p_2$, alors $\mathcal{L}(p) = \mathcal{L}(p_1) \times \mathcal{L}(p_2)$;*
- *si $p = x \mathbin{@} q$, alors $\mathcal{L}(p) = \mathcal{L}(q)$.*

En pratique, on peut ainsi ramener l'étude de la sémantique d'un motif non-linéaire à celle de sa forme linéarisée puisqu'on a une relation d'inclusion entre les deux :

Proposition 5.1. *Étant donné un motif étendu $p \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$, on a $[\![p]\!] \subseteq [\![\mathcal{L}(p)]\!]$.*

Démonstration. La preuve est directe par préservation de la sémantique du motif linéaire $\mathcal{L}(p)$ par substitution (Proposition 3.6) en considérant σ la substitution qui instancie chaque variable fraîche de $\mathcal{L}(p)$ par sa variable d'origine dans p. □

Afin de prouver qu'une règle préserve la sémantique, il faut cependant considérer la sémantique de l'instanciation de cette règle par substitution. On a ainsi fait le choix, dans le Chapitre précédent, de proposer une méthode pour l'analyse des systèmes linéaires, puisque comme on l'a montré dans la Section 3.2, la sémantique d'un terme non-linéaire n'est pas forcément préservée par substitution. L'approche par linéarisation permet cependant d'obtenir une telle préservation :

Corollaire 5.2. *Étant donné un motif étendu $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, pour toute substitution σ, on a $[\![\sigma(t)]\!] \subseteq [\![\mathcal{L}(t)]\!]$.*

Démonstration. La preuve est directe par préservation de la sémantique du motif linéaire $\mathcal{L}(p)$ par substitution (Proposition 3.6) en considérant σ' la substitution qui instancie chaque variable fraîche de $\mathcal{L}(p)$ par $\sigma(x)$ avec x sa variable d'origine dans p. □

On peut ainsi appliquer la méthode présentée dans le Chapitre précédent pour vérifier la condition suffisante à la satisfaction de profil de chaque règle du CBTRS donnée par le corollaire de la Proposition 4.15 suivant :

Corollaire 5.3. *Soit une règle $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$. On note $v := (\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!) \downarrow_{\mathfrak{R}}$, et on a :*

- *si $v = \bot$ alors la règle satisfait le profil π ;*
- *sinon, $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ avec chaque $\lambda_i^j, j \in [1, m]$ une version aliasée de $ls_i, i \in [1, n]$, et si $\forall k, \{\!|\mathcal{L}(\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs))|\!\} \cap [\![p]\!] = \emptyset$ alors la règle satisfait le profil π.*

Démonstration. La preuve est identique à celle de la Proposition 4.15. □

Exemple 5.1. *On considère l'algèbre des entiers de Peano définie par la signature $\Sigma = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par le type algébrique :*

$$\begin{array}{rcl} \mathsf{Int} & := & s(\mathsf{Int}) \\ & | & z \end{array}$$

et les symboles définis $\mathcal{D} = \{plus^{!\mathcal{P}_{plus}} : \mathsf{Int} * \mathsf{Int} \mapsto \mathsf{Int}, mult^{!\mathcal{P}_{mult}} : \mathsf{Int} * \mathsf{Int} \mapsto \mathsf{Int}\}$.

En considérant le CBTRS suivant, on veut prouver que la multiplication d'un entier quelconque avec 0 vaut 0 :

$$\begin{cases} plus(z, n) & \twoheadrightarrow & n \\ plus(s(i), n) & \twoheadrightarrow & s(plus(i, n)) \\ mult(z, n) & \twoheadrightarrow & z \\ mult(s(i), n) & \twoheadrightarrow & plus(n, mult(i, n)) \end{cases}$$

On considère donc les annotations suivantes : $\mathcal{P}_{plus} = \{s(i) * s(i) \mapsto s(i)\}$ pour exprimer que $0 + 0 = 0$, et $\mathcal{P}_{mult} = \{s(i) * \bot \mapsto s(i), \bot * s(i) \mapsto s(i)\}$ pour exprimer que $0 \times n = n \times 0 = 0$ pour tout entier n.

Les trois premières règles étant linéaires, on peut appliquer directement la méthode présentée dans le Chapitre précédent pour prouver qu'elles satisfassent les profils de leur symbole de tête. On s'intéresse de plus près à la dernière règle :

- *pour le profil $s(i) * \bot \mapsto s(i)$, on a $(\!|s(i) \times x_{\mathsf{Int}}^{-s(i)}, n \times y_{\mathsf{Int}}^{-\bot}|\!) \Longrightarrow_{\mathfrak{R}}^{*} \bot$, donc la règle satisfait le profil.*
- *pour le profil $\bot * s(i) \mapsto s(i)$, on a $(\!|s(i) \times x_{\mathsf{Int}}^{-\bot}, n \times y_{\mathsf{Int}}^{-s(i)}|\!) \Longrightarrow_{\mathfrak{R}}^{*} (\!|s(i \mathbin{@} z_{\mathsf{Int}}^{-\bot}), n \mathbin{@} y_{\mathsf{Int}}^{-s(i)}|\!)$. On note $u = \mathcal{L}(plus(n \mathbin{@} y_{\mathsf{Int}}^{-s(i)}, mult(i \mathbin{@} z_{\mathsf{Int}}^{-\bot}, n \mathbin{@} y_{\mathsf{Int}}^{-s(i)})))$, et il reste à vérifier que $\{\!|u|\!\} \cap [\![s(i)]\!] = \emptyset$. Par construction on a $u = plus(z_1{}_{\mathsf{Int}}^{-s(i)}, mult(z_2{}_{\mathsf{Int}}^{-\bot}, z_3{}_{\mathsf{Int}}^{-s(i)}))$, et la méthode présentée dans le Chapitre précédent permet de calculer son équivalent sémantique $\tilde{u} = x_{\mathsf{Int}}^{-s(i)}$. On a donc clairement $\{\!|\tilde{u}|\!\} \cap [\![s(i)]\!] = \emptyset$, ce qui peut être vérifié via l'algorithme `getReachable` et le système $\mathfrak{R}$.*

Toutes les règles satisfont donc bien tous les profils de leur symbole de tête, donc le système préserve la sémantique.

Dans l'exemple précédent, l'approche par linéarisation permet bien de vérifier la préservation de la sémantique parce que les contraintes imposées par la corrélation entre les instances de la variable n ne sont pas nécessaires pour vérifier la satisfaction des profils. En effet, la seule contrainte nécessaire est exprimée par l'étape d'inférence qui reconnaît que pour satisfaire le deuxième profil, les deux instances de n seront instanciées par un terme exempt de $s(i)$. Cependant, quand une contrainte de corrélation est indispensable pour pouvoir vérifier la satisfaction d'un profil, cette approche ne permettra pas de vérifier la préservation de la sémantique, ce qui dans ce contexte constituerait un faux-négatif.

Exemple 5.2. *On reprend l'algèbre de termes définie par la signature Σ_{nl} considérée dans l'Exemple 3.7 et on étudie le système $\mathcal{R}$ suivant :*

$$\begin{cases} f(c(x,y)) & \rightarrow & c(x,x) \\ f(d(x)) & \rightarrow & c(g(x),g(x)) \\ g(c(x,y)) & \rightarrow & a \\ g(d(x)) & \rightarrow & b \end{cases}$$

On rappelle que les symboles définis $f^{!\mathcal{P}_f}$ et $g^{!\mathcal{P}_g}$ sont annotés avec $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}$ et $\mathcal{P}_g = \emptyset$.

L'étude par linéarisation de la satisfaction du profil $\bot \mapsto c(a,b)$ par les deux premières règles revient à vérifier que $[\![c(x_{\mathsf{S1}}^{-\bot}, y_{\mathsf{S1}}^{-\bot})]\!] \cap [\![c(a,b)]\!] = \emptyset$ et $[\![c(g(x_{\mathsf{S1}}^{-\bot}), g(y_{\mathsf{S1}}^{-\bot}))]\!] \cap [\![c(a,b)]\!] = \emptyset$. Or on peut observer que $c(a,b) \in [\![c(x_{\mathsf{S1}}^{-\bot}, y_{\mathsf{S1}}^{-\bot})]\!] = [\![c(g(x_{\mathsf{S1}}^{-\bot}), g(y_{\mathsf{S1}}^{-\bot}))]\!]$. Cette approche ne permet donc pas de vérifier la préservation de la sémantique de ces deux règles, et ainsi ne peut pas garantir que la forme normale d'un terme de la forme $f(t)$ est exempte de $c(a,b)$.

Pour permettre l'étude de systèmes non-linéaires à partir de la méthode présentée dans le Chapitre précédent, on propose donc d'étudier comment le système $\mathfrak{R}$ peut être utilisé pour réduire une conjonction dont l'un des deux motifs n'est pas linéaire.

5.1.2 Étude de motifs non-linéaires

Pour un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on a défini dans le Chapitre 1 la relation de filtrage par motif qui établit qu'une valeur v est filtrée par p si et seulement il existe une substitution σ telle que $v = \sigma(t)$. Cette définition est valide même quand t n'est pas linéaire, mais pour les motifs étendus, la Définition 3.10 ne décrit que le filtrage par des motifs linéaires. Comme on veut présenter une application de la méthode d'analyse dans le cas non-linéaire, on commence par donner une définition générale du filtrage par motif :

Définition 5.2 (Filtrage par motif)**.** *Soient un motif étendu $p \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$ et une valeur $v \in \mathcal{T}(\mathcal{C})$, on dit que p filtre v, noté $p \prec\!\!\prec v$, si et seulement s'il existe une substitution σ telle que $p \overset{\sigma}{\prec\!\!\prec} v$ avec :*

$$\begin{array}{rcll} x_s^{-p} \overset{\sigma}{\prec\!\!\prec} v & \iff & v = \sigma(x_s^{-p}) \wedge v \text{ est exempt de } p & \text{pour } x_s^{-p} \in \mathcal{X}^a \\ c(p_1,\ldots,p_n) \overset{\sigma}{\prec\!\!\prec} c(v_1,\ldots,v_n) & \iff & \wedge_{i=1}^n p_i \overset{\sigma}{\prec\!\!\prec} v_i & \text{pour } c \in \mathcal{C} \\ p_1 + p_2 \overset{\sigma}{\prec\!\!\prec} v & \iff & p_1 \overset{\sigma}{\prec\!\!\prec} v \ \vee \ p_2 \overset{\sigma}{\prec\!\!\prec} v & \\ p_1 \setminus p_2 \overset{\sigma}{\prec\!\!\prec} v & \iff & p_1 \overset{\sigma}{\prec\!\!\prec} v \ \wedge \ p_2 \overset{\sigma}{\not\prec\!\!\prec} v & \\ p_1 \times p_2 \overset{\sigma}{\prec\!\!\prec} v & \iff & p_1 \overset{\sigma}{\prec\!\!\prec} v \ \wedge \ p_2 \overset{\sigma}{\prec\!\!\prec} v & \\ x \mathbin{@} p \overset{\sigma}{\prec\!\!\prec} v & \iff & x \times p \overset{\sigma}{\prec\!\!\prec} v & \end{array}$$

On a toujours $\bot \not\prec\!\!\prec v$ pour toute valeur $v \in \mathcal{T}(\mathcal{C})$.

Cette définition du filtrage par motif est équivalente à celle donnée dans le Chapitre 1 pour les motifs linéaires et permet de considérer la relation de filtrage par un motif étendu non-linéaire.

De même on peut ainsi définir la sémantique close d'un motif étendu :

Définition 5.3. *Étant donné un motif étendu $p \in \mathcal{P}(\mathcal{C}, \mathcal{X}^a)$, on définit la sémantique close de p, notée $[\![p]\!]$, par l'ensemble des valeurs filtrées par p :*

$$[\![p]\!] = \{v \in \mathcal{T}(\mathcal{C}) \mid p \preccurlyeq v\}$$

La méthode d'analyse proposée dans le Chapitre précédent repose sur trois mécanismes majeurs :

1. l'algorithme `getReachable`, et plus généralement la technique de décomposition de la sémantique profonde en sémantique close ;
2. le système de réduction $\mathfrak{R}$ permettant de calculer les conjonctions entre les termes de la décomposition et le motif considéré ;
3. un mécanisme d'aliasing permettant d'inférer les substitutions considérées par la notion de satisfaction de profil.

Sous certaines conditions (que l'on explicitera dans la Sous-section suivante), les mécanismes d'inférence et de décomposition de la sémantique profonde restent valides dans le cas non-linéaire. On propose donc d'étudier comment le système $\mathfrak{R}$ permet de réduire une conjonction dont l'un des deux motifs n'est pas linéaire.

On commence par observer que le système $\mathfrak{R}$ préserve également la sémantique des motifs non-linéaires [1] :

Proposition 5.4 (Préservation de la sémantique)**.** *Soient des motifs u et v, si $u \Longrightarrow^*_{\mathfrak{R}} v$ alors $[\![u]\!] = [\![v]\!]$.*

Démonstration. On prouve en Annexe A, en observant que le filtrage par motif est préservé par monotonie et par toutes les règles de $\mathfrak{R}$, qu'on a le Lemme suivant :

Lemme 5.5. *Soient p et q des motifs étendus conjonction-linéaires et linéaires à droite d'une conjonction, si $p \Longrightarrow_{\mathfrak{R}} q$, alors pour toute valeur $v \in \mathcal{T}(\mathcal{C})$, et pour toute substitution σ, on a :*

$$p \overset{\sigma}{\preccurlyeq} v \iff q \overset{\sigma}{\preccurlyeq} v$$

Soient u et v des motifs tels que $u \Longrightarrow^*_{\mathfrak{R}} v$. D'après le Lemme précédent, on a pour toute valeur $w \in \mathcal{T}(\mathcal{C})$, $u \preccurlyeq w$ si et seulement si $v \preccurlyeq w$. Donc par définition de la sémantique close, on a $[\![u]\!] = [\![v]\!]$. □

En pratique, le mécanisme d'aliasing, utilisé pour inférer les contraintes imposées sur les variables de la règle de réécriture par le membre gauche du profil considéré, permet également d'exprimer les contraintes de corrélation des différentes instances d'une même variable. On observe, en effet, qu'une version aliassée d'un terme issu du membre gauche de la règle a une sémantique vide, si et seulement si il existe une variable dont les contraintes d'instanciation ne sont pas satisfaisables :

Lemme 5.6. *Soit τ une version aliassée d'un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on a $[\![\tau]\!] = \emptyset$ si et seulement si il existe une variable $x \in \mathcal{V}ar(t)$ telle que $[\![\tau_{@x}]\!] = \emptyset$.*

1. On ne s'intéresse toujours qu'à des motifs étendus conjonction-linéaires et linéaires à droite d'une conjonction

Démonstration. Soit τ une version aliassée d'un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, on prouve les deux implications séparément.

On suppose que $[\![\tau]\!] \neq \emptyset$. Il existe donc $v \in [\![\tau]\!]$, et il existe ainsi une substitution σ telle que $\tau \overset{\sigma}{\lessdot} v$. Donc pour toute variable $x \in \mathcal{V}ar(t)$, pour toute position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = x$, on a $\tau_{|\omega} = x \mathbin{@} q$ et $q \overset{\sigma}{\lessdot} \sigma(x)$. Donc, on a bien pour toute variable $x \in \mathcal{V}ar(t)$, $\tau_{@x} \overset{\sigma}{\lessdot} \sigma(x)$ d'où $\sigma(x) \in [\![\tau_{@x}]\!]$.

On suppose que pour toute variable $x \in \mathcal{V}ar(t)$, on a $[\![\tau_{@x}]\!] \neq \emptyset$. Donc, pour chaque variable $x \in \mathcal{V}ar(t)$, il existe $v_x \in [\![\tau_{@x}]\!]$ et il existe ainsi une substitution σ_x telle que $\tau_{@x} \overset{\sigma_x}{\lessdot} v_x$. Comme toute variable $y \in \mathcal{MV}(\tau_{@x})$ distincte de x n'apparaît qu'une fois dans τ, on peut définir $\sigma := \{y \mapsto \sigma_x(y) \mid x \in \mathcal{V}ar(t), y \in \mathcal{MV}(\tau_{@x})\}$. On a alors $\tau \overset{\sigma}{\lessdot} \sigma(t)$, d'où $\sigma(t) \in [\![\tau]\!]$. □

Le système $\mathfrak{R}$ permet alors de réduire correctement la conjonction considérée :

Proposition 5.7. *Étant donnés un motif quasi-symbolique t et un motif additif linéaire et régulier p, on note $v := (t \times p) \downarrow_{\mathfrak{R}}$ et on a $[\![t \times p]\!] = \emptyset$ si et seulement si*

- *$v = \bot$, ou*
- *$v = \tau_1 + \cdots + \tau_n$, avec $\tau_i, i \in [1, n]$, une version aliasée de t telle qu'il existe une variable $x \in \mathcal{V}ar(t)$ avec $\tau_{i@x} \Longrightarrow^*_{\mathfrak{R}} \bot$.*

Démonstration. D'après le Lemme 4.14, on a $v = \bot$ ou $v = \tau_1 + \cdots + \tau_n$, avec $\tau_i, i \in [1, n]$, une version aliasée de t. De plus, d'après la Proposition 5.4, on a $[\![\tau \times p]\!] = [\![v]\!]$.

Par conséquent, si $v = \bot$, alors $[\![\tau \times p]\!] = \emptyset$. Sinon, on a :

$$\begin{array}{rcl} [\![\tau \times q]\!] = \emptyset & \iff & \forall i \in [1, n], [\![\tau_i]\!] = \emptyset \\ & \iff & \forall i \in [1, n], \exists\, x \in \mathcal{V}ar(r), [\![\tau_{i@x}]\!] = \emptyset \\ & \iff & \forall i \in [1, n], \exists\, x \in \mathcal{V}ar(r), \tau_{i@x} \Longrightarrow^*_{\mathfrak{R}} \bot \end{array}$$

□

On peut ainsi vérifier des propriétés d'Exemption de Motif pour des motifs non-linéaires.

Exemple 5.3. *On reprend l'algèbre de termes définie par la signature Σ_{nl} considérée dans l'Exemple 3.7 et on prouve que le terme $c(x, x)$ est exempt de $c(a, b)$. D'après la Proposition 3.9, on cherche à vérifier que $\{\!\{c(x, x)\}\!\} \cap [\![c(a, b)]\!] = \emptyset$. On a $\{\!\{c(x, x)\}\!\} = [\![c(x, x)]\!] \cup [\![x^{-\bot}_{\mathsf{S2}}]\!]$. Comme $c(a, b)$ est de sorte $\mathsf{S1}$, on évalue la forme normale v de la conjonction $c(x^{-\bot}_{\mathsf{S2}}, x^{-\bot}_{\mathsf{S2}}) \times c(a, b)$:*

$$\begin{array}{rcl} c(x^{-\bot}_{\mathsf{S2}}, x^{-\bot}_{\mathsf{S2}}) \times c(a, b) & \xrightarrow{T3} & c(x^{-\bot}_{\mathsf{S2}} \times a, x^{-\bot}_{\mathsf{S2}} \times b) \\ & \xrightarrow{T2}_{\twoheadrightarrow} & c(x^{-\bot}_{\mathsf{S2}} \mathbin{@} a, x^{-\bot}_{\mathsf{S2}} \mathbin{@} b) = v \end{array}$$

De plus, on a $v_{@x} = a \times b \xrightarrow{T4} \bot$, d'où $c(x, x)$ est bien exempt de $c(a, b)$.

5.1.3 Stricte préservation

Afin de prouver la préservation de la sémantique (et, par extension, des propriétés d'Exemption de Motif) par réécriture, on a proposé dans le Chapitre 3 de considérer les notions de règles préservant la sémantique (Définition 3.14) et/ou satisfaisant un profil (Définition 3.15). La préservation des notions d'Exemption de Motif et de sémantique n'étant pas garantie pour les termes non-linéaires, ces propriétés ne sont pas adaptées à l'analyse d'un système non-linéaire.

Néanmoins, on a montré dans la Proposition 3.19, que les notions d'Exemption de Motif et de sémantique sont préservées par substitution valeur. On peut ainsi reformuler les notions de préservation de sémantique (Définition 3.14) et de satisfaction de profil (Définition 3.15) dans le cadre d'une relation de réécriture basée sur les substitutions valeurs (*i.e.* une relation de réécriture avec une stratégie de réduction stricte).

Définition 5.4 (Préservation stricte de sémantique)**.** *Soit une règle de réécriture $ls \twoheadrightarrow rs$, on dit que la règle préserve strictement la sémantique si et seulement si, pour toute substitution valeur ς, on a $[\![\varsigma(rs)]\!] \subseteq [\![\varsigma(ls)]\!]$.*

On dit qu'un TRS $\mathcal{R}$ préserve strictement la sémantique si et seulement si toutes les règles de $\mathcal{R}$ préservent strictement la sémantique.

Comme pour la préservation classique (Définition 3.14), cette notion de préservation revient à considérer l'ensemble des applications en tête possibles d'une des règles du TRS. De façon similaire, un TRS préservant strictement la sémantique induit une relation de réécriture qui préserve également la sémantique dans le cadre d'une stratégie de réduction stricte (cf. Définition 1.29) [1] :

Proposition 5.8. *Soit un TRS $\mathcal{R}$ préservant strictement la sémantique, on a pour tout termes clos $t, v \in \mathcal{T}(\mathcal{F})$:*

$$t \longrightarrow_{\mathcal{R}}^{*} v \implies [\![v]\!] \subseteq [\![t]\!]$$

Démonstration. La preuve est identique à celle de la Proposition 3.15, en ne considérant ici que des substitutions valeurs. □

En pratique, l'étude d'une telle relation de réécriture est néanmoins plus limitante puisque cela revient à considérer une stratégie d'évaluation en *Appel par valeur*, ce qui n'est pas forcément la stratégie universellement utilisée par tous les langages de programmation.

De même, on définit une notion de satisfaction stricte de profil :

Définition 5.5 (Satisfaction stricte de profil)**.** *Soit une règle de réécriture $\varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$, on dit que la règle satisfait strictement le profil π si et seulement pour toute substitution valeur ς, on a :*

$$\varsigma(ls_i) \text{ est exempt de } p_i, \text{ pour tout } i \in [1, n] \implies \varsigma(rs) \text{ est exempt de } p$$

Et on a également une équivalence entre la satisfaction stricte de tous les profils d'une règle et sa préservation stricte de la sémantique :

Proposition 5.9. *Soit une règle de réécriture $\varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$, la règle préserve strictement la sémantique si et seulement elle satisfait strictement tous les profils de $\mathcal{P}$.*

Démonstration. La preuve est identique à celle de la Proposition 3.18. □

Contrairement à l'approche par linéarisation, on peut ainsi utiliser la méthode présentée dans la Sous-section précédente pour la réduction, en utilisant le mécanisme d'aliasing pour prendre en compte les contraintes de corrélation entre plusieurs instances d'une même variable. Cela permet donc une analyse plus précise du comportement d'un système non-linéaire.

1. On rappelle que la notation $\longrightarrow_{\mathcal{R}}$ représente la relation de réécriture stricte induite d'un système $\mathcal{R}$, et $\longrightarrow_{\mathcal{R}}^{*}$ sa fermeture reflexive et transitive.

Corollaire 5.10. *Soit une règle* $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$*, et un profil* $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$*. On note* $v := (\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!) \downarrow_{\mathfrak{R}}$*, et on a :*

- *si* $v = \bot$ *alors la règle satisfait strictement le profil* π *;*
- *sinon,* $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ *avec chaque* $\lambda_i^j, j \in [1, m]$ *une version aliasée de* $ls_i, i \in [1, n]$*, et si* $\forall k, [\![\sigma^{@(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)}_{(\!|ls_1, \ldots, ls_n|\!)}(rs)]\!] \cap [\![p]\!] = \emptyset$ *alors la règle satisfait strictement le profil* π*.*

Démonstration. La preuve est identique à celle de la Proposition 4.15. □

Exemple 5.4. *On reprend l'algèbre de termes définie par la signature* Σ_{nl} *considérée dans l'Exemple 3.7 et on étudie le système* $\mathcal{R}$ *suivant*

$$\begin{cases} f(c(x,a)) & \twoheadrightarrow & c(x,x) \\ f(c(x,b)) & \twoheadrightarrow & h(x,x) \\ f(d(x)) & \twoheadrightarrow & c(g(x),g(x)) \\ g(c(x,y)) & \twoheadrightarrow & a \\ g(d(x)) & \twoheadrightarrow & b \\ h(x,y) & \twoheadrightarrow & c(y,x) \end{cases}$$

avec les symboles définis $f^{!\mathcal{P}_f}, g^{!\mathcal{P}_g}$ *et* $h^{!\mathcal{P}_h}$ *sont annotés avec* $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}, \mathcal{P}_g = \emptyset$ *et* $\mathcal{P}_h = \{b * \bot \mapsto c(a,b), \bot * a \mapsto c(a,b)\}$*.*

Contrairement à l'approche par linéarisation, on peut vérifier la stricte satisfaction de la règle $f^{!\mathcal{P}_f}(c(x,a)) \twoheadrightarrow c(x,x)$*. On a en effet* $(c(x_{\mathsf{S2}}^{-\bot}, a) \times z_{\mathsf{S1}}^{-\bot}) \downarrow_{\mathfrak{R}} = c(x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot}, a)$*, et en notant* $u := c(x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot}, x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot})$*, on doit donc vérifier que* $[\![u]\!] \cap [\![c(a,b)]\!] = \emptyset$*. Comme dans l'Exemple 5.3, on a* $[\![u]\!] = [\![u]\!] \cup [\![x_{\mathsf{S2}}^{-\bot}]\!]$*, et* $v := (u \times c(a,b)) \downarrow_{\mathfrak{R}} = c(x \mathbin{@} a, y \mathbin{@} b)$*. Donc* $v_{@x} = a \times b \xrightarrow{T4} \bot$ *et* $[\![u]\!] \cap [\![c(a,b)]\!] = \emptyset$*.*

En revanche, le formalisme d'Exemption de Motif (et de sémantique) utilisé ignore la corrélation entre les deux instances de x *dans le membre droit de la règle* $f^{!\mathcal{P}_f}(c(x,b)) \twoheadrightarrow h(x,x)$ *: x ne peut être instancié que par a ou b, dans le premier cas* $\sigma(x)$ *est exempt de b et* $\sigma(h(x,x))$ *est exempt de* $c(a,b)$ *par le profil* $b * \bot \mapsto c(a,b)$*, et dans le deuxième cas* $\sigma(x)$ *est exempt de a et* $\sigma(h(x,x))$ *est exempt de* $c(a,b)$ *par le profil* $\bot * a \mapsto c(a,b)$*. Cependant, si on essaie de vérifier la stricte satisfaction, on a* $(c(x_{\mathsf{S2}}^{-\bot}, b) \times z_{\mathsf{S1}}^{-\bot}) \downarrow_{\mathfrak{R}} = c(x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot}, b)$*, et en notant* $u := c(x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot}, x \mathbin{@} z_1 {}_{\mathsf{S2}}^{-\bot})$*, on doit donc vérifier que* $[\![u]\!] \cap [\![c(a,b)]\!] = \emptyset$*. De plus, on a* $\tilde{u} = x_{\mathsf{S1}}^{-\bot}$ *puisqu'aucun des deux profils de* $\mathcal{P}_h$ *ne s'applique. Par conséquent, on a* $[\![u]\!] \cap [\![c(a,b)]\!] = \{c(a,b)\}$ *et la stricte satisfaction ne peut pas être vérifiée.*

Enfin, on peut observer que pour la règle $f(d(x)) \twoheadrightarrow c(g(x),g(x))$ *la corrélation entre les deux instances de la variable x dans le membre droit est ignorée par le formalisme d'Exemption de Motif (et de sémantique) parce qu'elles se trouvent sous des symboles définis. Cependant comme les deux sous-termes en question* $g(x)$ *et* $g(x)$ *sont identiques, on peut cependant argumenter qu'étant une relation de réécriture confluente, ou appliquée avec une stratégie déterministe, ces deux instances seront nécessairement réduits de façon identique, i.e. la première ne peut pas être réduite à a et la deuxième à b. Dans ce contexte, toute potentielle forme normale de* $c(g(x),g(x))$ *serait donc bien exempte de* $c(a,b)$*.*

Cette approche est donc non seulement limitée par la supposition d'une relation de réécriture stricte, mais de plus, elle ne permet pas toujours de prendre en compte toutes les contraintes de corrélation entre différentes instances d'une même variable. Bien que cela corresponde généralement à des cas très particuliers, on propose dans la Section suivante une adaptation du formalisme d'Exemption de Motif et de sémantique permettant de considérer de façon plus précise ces contraintes, sans dépendre d'une stratégie de réécriture stricte.

5.2 Formalisme non-linéaire

Dans le Chapitre précédent, on a présenté une méthode d'analyse permettant de vérifier que la relation de réécriture induite par un CBTRS donné préserve la sémantique. Cette méthode se base sur la notion de sémantique close d'un terme qui est utilisée pour représenter une sur-approximation des formes normales potentielles des termes considérés. Une telle méthode basée sur la construction et l'évaluation de sur-approximations peut naturellement mener à des résultats négatifs en désaccord avec le comportement réel de la transformation étudiée, *i.e.* des *faux-négatifs*. C'est tout particulièrement le cas pour l'étude de systèmes non-linéaires puisque les approches considérées pour appliquer la méthode sur de tels systèmes (présentées dans la Section précédente) présentent certaines limitations :

- la première fonctionne par linéarisation, on perd ainsi toute information potentiellement imposée par les contraintes de corrélation entre les variables. La sur-approximation considérée est alors plus grossière que précédemment comme noté par le fait que l'on a qu'une condition suffisante de la préservation de la sémantique (Proposition 5.3).
- la deuxième propose d'étudier le comportement des termes constructeurs non-linéaires, ce qui permet de prendre en compte certaines de ces contraintes. En pratique, on a encore de nombreux cas où cette approche ne permet pas de prendre en compte toutes les contraintes de corrélation des variables. De plus, bien que cela permette une étude moins grossière de la sémantique, comme la sémantique close d'un terme non-linéaire n'est préservée que par des substitutions valeurs, cela limite également la méthode à l'étude de relation de réécriture stricte.

Comme on l'a vu dans la Section précédente, l'approche par sémantique sur laquelle se fonde la méthode d'analyse est capable d'exprimer les contraintes de corrélation entre différentes instances d'une même variable dans un terme constructeur. Cependant les formalismes d'Exemption de Motif et de sémantique ignorent la majorité de ces contraintes dans le cas d'un terme contenant des symboles définis. Enfin, prouver la préservation de la sémantique d'un CBTRS est d'autant plus compliqué que ces notions ne sont pas préservées par substitution.

On introduit dans cette Section des notions adaptées de sémantique et d'Exemption de Motif permettant de prendre en compte de manière plus précise les contraintes de corrélation entre différentes instances d'une même variable.

5.2.1 Exemption de Motif pour les termes non-linéaires

Dans le Chapitre 3, on a défini les notions d'Exemption de Motif et de sémantique close de façon à avoir une équivalence simple entre ces deux notions, comme exprimé par la Proposition 3.7. On va présenter ici différentes formes de sémantique afin de les comparer dans le contexte de la méthode d'analyse présentée dans le Chapitre précédent. Pour garder une équivalence entre ces différentes sémantiques et des notions associées d'Exemption de Motif, on définit une Exemption de Motif modulo sémantique :

Définition 5.6. *Étant donnés un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, un motif additif p et une sémantique $[\![]\!]$, on dit que t est exempt de p modulo $[\![]\!]$ si et seulement si, pour tout $v \in [\![t]\!]$, v est exempt de p.*

Par la suite, pour la distinguer des autres formes de sémantique, on note $[\![t]\!]_g$ la sémantique close d'un terme t (Définition 3.4). D'après la Proposition 3.7, qui établit une équivalence entre l'Exemption de Motif pour un terme t et l'Exemption de Motif de toutes les valeurs de sa sémantique, l'Exemption de Motif classique est donc équivalente à l'Exemption de Motif modulo $[\![]\!]_g$.

Comme montré dans le Chapitre 3, dans le cas non-linéaire, l'absence de préservation de l'Exemption de Motif et de sémantique par substitution s'explique par le fait que ces notions ne considèrent aucune corrélation entre deux sous-termes identiques ayant un symbole défini comme symbole de tête. En effet, ces notions sont définies en sur-approximant chaque instance d'un tel sous-terme séparément par l'ensemble de ses potentielles formes normales. Quand une substitution associe une variable à un tel terme, la contrainte de corrélation entre deux instances de cette variable est donc brisée par cette approche.

Cependant, dans le cadre de l'étude de transformations, ces dernières sont généralement appliquées de manière déterministe, donc deux termes identiques seront réduits de manière identique. Du point de vue de la réécriture, cela revient à considérer une hypothèse de confluence de la relation de réécriture, ou l'utilisation d'une stratégie de réécriture déterministe. Dans ce cas, on propose de considérer ce comportement déterministe en introduisant, pour des termes t, u et $v \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ la notation $t\,[u \mapsto v]$ pour désigner le terme t dont toutes les occurrences du terme u sont remplacées par v :

$$t\,[u \mapsto v] = t\,[v]_{\omega_1} \cdots [v]_{\omega_n} \qquad \text{avec } \{\omega_1, \ldots, \omega_n\} := \{\omega \in \mathcal{P}os(t) \mid t_{|\omega} = u\}$$

On introduit maintenant différentes formes de sémantique :

Définition 5.7 (Sémantiques non-linéaires)**.** *On définit les notions de sémantiques suivantes :*

- *étant donnés un terme $u \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ et un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, la sémantique confluente de u, respectivement t, est définie par :*
  - $[\![u]\!]_c = [\![u]\!]_g$;
  - $[\![t]\!]_c = \bigcup_\omega \bigcup_v [\![t\,[t_{|\omega} \mapsto v]]\!]_c$, *avec $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$.*

  où $\triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n) = \sum_{q \in \mathcal{Q}} q$ avec $\mathcal{Q} = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$ t.q. $\forall i \in [1, n], \forall v \in [\![t_i]\!]_c, v$ est exempt de $l_i\}$.
- *étant donnés un terme $u \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ et un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, la sémantique modulo ς de u, respectivement t, avec ς une substitution valeur telle que $\mathcal{V}ar(u) \subseteq \mathcal{D}om(\varsigma)$, respectivement $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$, et pour toute variable $x_s^{-q} \in \mathcal{D}om(\varsigma)$, $\varsigma(x_s^{-q})$ est exempt de q, est définie par :*
  - $[\![u]\!]_\varsigma = \{\varsigma(u)\}$;
  - $[\![t]\!]_\varsigma = \bigcup_\omega \bigcup_{\varsigma'} [\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_{\varsigma'}$, *avec $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$ et pour tout $i \in [1, n]$ $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, z_s^{-p} une variable fraîche avec $p = \triangleright^{\mathcal{P}}(\varsigma(t_1), \ldots, \varsigma(t_n))$, et ς' une substitution valeur telle que $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$ et $\varsigma'(z_s^{-p}) \in \mathcal{T}_s(\mathcal{C})$ est exempt de p.*
- *étant donné un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$, la sémantique substitutive de t est définie par :*
  - $[\![t]\!]_s = \bigcup_\varsigma [\![t]\!]_\varsigma$ *avec ς une substitution valeur telle que $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$ et pour toute variable $x_s^{-q} \in \mathcal{D}om(\varsigma)$, $\varsigma(x_s^{-q})$ est exempt de q.*
- *étant donnés une valeur $v \in \mathcal{T}(\mathcal{C})$, un terme $u \in \mathcal{T}(\mathcal{F})$ et un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{F})$, la sémantique exacte de v, respectivement u et t, est définie par :*
  - $[\![v]\!]_e = \{v\}$;
  - $[\![u]\!]_e = \bigcup_\omega \bigcup_v [\![u\,[u_{|\omega} \mapsto v]]\!]_e$, *avec $\omega \in \mathcal{P}os(u)$ telle que $u_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$ et pour tout $i \in [1, n]$ $u_i \in \mathcal{T}(\mathcal{C})$, et $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$;*

- $[\![t]\!]_e = \bigcup_\varsigma [\![\varsigma(t)]\!]_e$ *avec* ς *une substitution valeur telle que pour toute variable* $x_s^{-p} \in \mathcal{V}ar(t), \varsigma(x_s^{-p})$ *est une valeur exempte de* p.

La notion de sémantique confluente correspond à la sémantique close en considérant l'hypothèse de confluence : la sur-approximation des sous-termes ayant un symbole défini comme symbole de tête est appliquée de façon identique pour chaque instance du sous-terme. La notion de sémantique exacte correspond à la sur-approximation obtenue en évaluant récursivement du bas vers le haut les sous-termes et en considérant toutes les corrélations entre les différentes instances d'une même variable et les différentes occurrences de sous-termes identiques via une approche par substitution. La notion de sémantique substitutive exprime une sur-approximation intermédiaire entre les deux dernières en considérant une approche substitutive permettant de prendre en compte les corrélations entre les différentes instances d'une même variable mais en ignorant les contraintes de corrélation entre sous-termes issus de la substitution considérée.

Puisque l'hypothèse de confluence permet à ces trois notions de sémantique de considérer les contraintes de corrélation entre les différentes occurrences d'un sous-terme ayant un symbole défini comme symbole de tête, ces sémantiques sont correctement préservées par substitution :

Proposition 5.11. *Soient un terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ *et une substitution* σ*, on a :*

- $[\![\sigma(t)]\!]_c \subseteq [\![t]\!]_c$ *;*
- $[\![\sigma(t)]\!]_s \subseteq [\![t]\!]_s$ *;*
- $[\![\sigma(t)]\!]_e \subseteq [\![t]\!]_e$.

Démonstration. On prouve la première inclusion par induction sur k, le nombre de symboles définis dans $\sigma(t)$:

- Si $k = 0$, alors $\sigma(t) \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ et $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$. On considère $w \in [\![\sigma(t)]\!]_c$, par définition, il existe donc une substitution valeur ς telle que $w = \varsigma(\sigma(t))$. D'où $w = \varsigma \circ \sigma(t) \in [\![t]\!]_c$.
- On considère maintenant $k > 0$ tel que pour tout terme q et substitution σ', si $\sigma'(q)$ a strictement moins de k symboles définis, alors $[\![\sigma'(q)]\!]_c \subseteq [\![q]\!]_c$. On note $u = \sigma(t)$, et on considère $w \in [\![u]\!]_c$. Par définition, il existe une position $\omega \in \mathcal{P}os(u)$ où $u_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright_c^{\mathcal{P}}(u_1, \ldots, u_n)$, telles que $w \in [\![u\left[u_{|\omega} \mapsto v\right]]\!]_c$.
  - S'il existe au moins une position $\omega' \in \mathcal{P}os(t)$ telle que $t_{|\omega'} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ et $\sigma(t_i) = u_i$ pour tout $i \in [1, n]$, on note $t' = t\left[t_{|\omega'} \mapsto v\right]$. On a alors $\sigma(t')\left[u_{|\omega} \mapsto v\right] = u\left[u_{|\omega} \mapsto v\right]$, d'où $w \in [\![\sigma(t')]\!]_c$. D'après l'hypothèse d'induction, on a donc $w \in [\![t']\!]_c$ et $[\![\sigma(t_i)]\!]_c \subseteq [\![t_i]\!]_c$. Par conséquent v est exempt de $\triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$, et on a bien $w \in [\![t']\!]_c \subseteq [\![t]\!]_c$.
  - Sinon, on définit $\sigma' = \{x \mapsto \sigma(x)\left[u_{|\omega} \mapsto v\right] \mid x \in \mathcal{D}om(\sigma)\}$. On a alors $u\left[u_{|\omega} \mapsto v\right] = \sigma'(t)$, et d'après l'hypothèse d'induction, on a donc $w \in [\![t]\!]_c$.

Pour la deuxième inclusion, le même raisonnement permet de prouver par induction sur k le nombre de symboles définis dans $\sigma(t)$, le Lemme suivant :

Lemme 5.12. *Soient un terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ *et une substitution* σ*, on a, pour toute substitution valeur* ς *avec* $\mathcal{V}ar(\sigma(t)) \subseteq \mathcal{D}om(\varsigma)$*,* $[\![\sigma(t)]\!]_\varsigma \subseteq \bigcup_{\varsigma'} [\![t]\!]_{\varsigma'}$ *avec* ς' *une substitution valeur telle que*

- $\varsigma'(x) \in [\![\sigma(x)]\!]_\varsigma$ *pour toute variable* $x \in \mathcal{D}om(\sigma)$ *;*
- $\varsigma'(x) = \varsigma(x)$ *pour toute variable* $x \in \mathcal{D}om(\varsigma) \setminus \mathcal{D}om(\sigma)$.

D'où $[\![\sigma(t)]\!]_s \subseteq [\![t]\!]_s$.

Enfin, on prouve la dernière inclusion par induction sur k le nombre de symboles définis dans $\sigma(t)$:

- Si $k = 0$, alors $\sigma(t) \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ et $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$. On a trois cas possibles :
  - Si $\sigma(t) \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{C})$, alors $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{C})$ et pour tout $v \in [\![\sigma(t)]\!]_e$, il existe ς telle que $\varsigma(\sigma((t)) = v$. D'où $v \in [\![t]\!]_e$.
  - Si $t \in \mathcal{T}(\mathcal{C})$, alors $\sigma(t) = t$ et $[\![t]\!]_e = [\![\sigma(t)]\!]_e = \{t\}$.
  - Enfin, si $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a) \setminus \mathcal{T}(\mathcal{C})$ et $\sigma(t) \in \mathcal{T}(\mathcal{C})$, alors $\sigma(t) \in [\![t]\!]_e$. D'où $[\![\sigma(t)]\!]_e = \{\sigma(t)\} \subseteq [\![t]\!]_e$.
- On considère maintenant $k > 0$ tel que pour tout terme q et substitution σ', si $\sigma'(q)$ a strictement moins de k symboles définis, alors $[\![\sigma'(q)]\!]_e \subseteq [\![q]\!]_e$. On considère $w \in [\![\sigma(t)]\!]_c$. Si $\sigma(t) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{T}(\mathcal{F})$, il existe une substitution valeur ς telle que $v \in [\![\varsigma(\sigma((t))]\!]_e$, et on note $u = \varsigma(\sigma(t))$ et $\sigma_u = \varsigma \circ \sigma$. Sinon on note $u = \sigma(t)$ et $\sigma_u = \sigma$. Dans les deux cas, il existe une position $\omega \in \mathcal{P}os(u)$ où $u_{|\omega} = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$ et $u_i \in \mathcal{T}(\mathcal{C}), \forall i \in [1, n]$, et $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(u_1, \ldots, u_n)$, telles que $w \in [\![u\left[u_{|\omega} \mapsto v\right]]\!]_e$. On définit $\sigma' = \{x \mapsto \sigma_u(x)\left[u_{|\omega} \mapsto v\right] \mid x \in \mathcal{D}om(\sigma_u)\}$ et on a deux cas à considérer :
  - Si $\sigma' = \sigma_u$, on note $u = u\left[u_{|\omega} \mapsto v\right]$ et on recommence l'opération jusqu'à ce que $\sigma' \neq \sigma_u$ ou que $u \in \mathcal{T}(\mathcal{C})$ (au plus k fois). Dans le deuxième cas, on a donc σ_u valeur avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\sigma_u)$, d'où $v \in [\![t]\!]_e$.
  - Si $\sigma' \neq \sigma_u$, $\sigma'(t)$ a strictement moins de k symboles définis et comme $u\left[u_{|\omega} \mapsto v\right] = \sigma'(t)\left[u_{|\omega} \mapsto v\right]$, $v \in [\![\sigma'(t)\left[u_{|\omega} \mapsto v\right]]\!]_e \subseteq [\![\sigma'(t)]\!]_e$. Donc, d'après l'hypothèse d'induction, $v \in [\![t]\!]_e$.

□

De même, les propriétés d'Exemption de Motif modulo sémantique sont également préservée par substitution :

Corollaire 5.13. *Soient un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ et un motif $p \in \mathcal{P}(\mathcal{C}, \mathcal{X})$, on a :*

- *si t est exempt de p modulo $[\![]\!]_c$ alors, pour toute substitution σ, $\sigma(t)$ est exempt de p modulo $[\![]\!]_c$;*
- *si t est exempt de p modulo $[\![]\!]_s$ alors, pour toute substitution σ, $\sigma(t)$ est exempt de p modulo $[\![]\!]_s$;*
- *si t est exempt de p modulo $[\![]\!]_e$ alors, pour toute substitution σ, $\sigma(t)$ est exempt de p modulo $[\![]\!]_e$.*

Démonstration. La preuve est immédiate en utilisant la Proposition précédente et la Définition de l'Exemption de Motif modulo sémantique. □

On s'intéresse dans un premier temps à la sémantique confluente. Bien que cette sémantique permette de garantir la préservation par substitution et de considérer les contraintes de corrélation entre différentes instances d'une même variable se trouvant sous différentes occurrences d'un même symbole défini, elle ignore toujours ces contraintes de corrélation quand ces instances d'une même variable se trouvent dans différents sous-termes d'un symbole défini.

Exemple 5.5. *On reprend l'algèbre de termes considérée dans l'Exemple 3.7 avec les symboles définis $f^{!\mathcal{P}_f}, g^{!\mathcal{P}_g}, h^{!\mathcal{P}_h}$ et $id_{\mathsf{S2}}^{!\mathcal{P}_{id}}$ annotés par $\mathcal{P}_f = \{\bot \mapsto c(a, b)\}, \mathcal{P}_g = \emptyset, \mathcal{P}_h = \{b * \bot \mapsto c(a, b), \bot * a \mapsto c(a, b)\}$ et $\mathcal{P}_{id} = \{b \mapsto b\}$. On considère des propriétés d'Exemption de Motif modulo $[\![]\!]_c$:*

- *$t = c(g(x), g(x))$ est exempt de $c(a, b)$ modulo $[\![]\!]_c$, puisque par définition de la sémantique confluente, on a $[\![t]\!]_c = [\![t\left[g(x) \mapsto a\right]]\!]_c \cup [\![t\left[g(x) \mapsto b\right]]\!]_c = \{c(a, a), c(b, b)\}$.*

- $u = h(x,x)$ *n'est pas exempt de* $c(a,b)$ *modulo* $[\![]\!]_c$*, puisque* $[\![x_{\mathsf{S2}}^{-\bot}]\!]_c = \{a,b\}$*, donc on a* $\triangleright_c^{\mathcal{P}_h}(x,x) = \bot$*. Par conséquent,* $[\![u]\!]_c = \bigcup_{v \in \mathcal{T}_{\mathsf{S1}}(\mathcal{C})} [\![u\,[u \mapsto v]]\!]_c = \mathcal{T}_{\mathsf{S1}}(\mathcal{C})$*. Néanmoins, comme* x *ne peut être instancié que par* a *ou* b*, dans le premier cas* $\sigma(x)$ *est exempt de* b *et* $\sigma(u)$ *est exempt de* $c(a,b)$ *par le profil* $b * \bot \mapsto c(a,b)$*, et dans le deuxième cas* $\sigma(x)$ *est exempt de* a *et* $\sigma(u)$ *est exempt de* $c(a,b)$ *par le profil* $\bot * a \mapsto c(a,b)$*.*
- $r = c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(x))$ *n'est pas exempt de* $c(a,b)$ *modulo* $[\![x_{\mathsf{S2}}^{-\bot}]\!]_c = \{a,b\}$*, donc on a* $\triangleright_c^{\mathcal{P}_{id}}(x) = \bot$*. Et comme* $\triangleright_c^{\mathcal{P}_{id}}(b) = \bot$*, on a* $[\![r]\!]_c = \{c(a,a), c(a,b), c(b,a), c(b,b)\}$*. Néanmoins, comme* x *ne peut être instancié que par* a *ou* b*, dans le premier cas* $\sigma(x)$ *est exempt de* b *et* $id(\sigma(x))$ *est exempt de* b *par le profil* $b \mapsto b$*, et dans le deuxième cas* $\sigma(id(x)) = \sigma(id(a))$*. Dans les deux cas, on a donc* $\sigma(r)$ *exempt de* $c(a,b)$*.*

A l'opposé, on a la notion de sémantique exacte, qui fournit une sur-approximation des formes normales potentielles en considérant toutes les instances possibles du terme et en évaluant récursivement, du bas vers le haut, la sur-approximation des sous-termes ayant un symbole défini comme symbole de tête. Cette approche permet de prendre parfaitement en compte l'ensemble des contraintes de corrélation des différentes instances d'une même variable tout en considérant les corrélations entre différentes occurrences d'un même sous-terme.

Exemple 5.6. *On reprend l'algèbre de termes considérée dans l'Exemple 5.5. On considère des propriétés d'Exemption de Motif modulo* $[\![]\!]_e$ *:*

- $t = c(g(x), g(x))$ *est exempt de* $c(a,b)$ *modulo* $[\![]\!]_e$*, puisque pour toute substitution* σ *on a* $[\![\sigma(t)]\!]_e = [\![\sigma(t)\,[g(\sigma(x)) \mapsto a]]\!]_e \cup [\![t\,[g(\sigma(x)) \mapsto b]]\!]_c = \{c(a,a), c(b,b)\}$*. Donc* $[\![t]\!]_e = \{c(a,a), c(b,b)\}$*.*
- $u = h(x,x)$ *est exempt de* $c(a,b)$ *modulo* $[\![]\!]_e$*, puisque* x *ne peut être instancié que par* a *ou* b*, donc* $[\![u]\!]_e = [\![h(a,a)]\!]_e \cup [\![h(b,b)]\!]_e$*. De plus* $[\![h(a,a)]\!]_e = [\![h(b,b)]\!]_e = \{v \in \mathcal{T}_{\mathsf{S1}}(\mathcal{C}) \mid v \text{ exempt de } c(a,b)\}$*, car* $\triangleright^{a,a}(\mathcal{P}_h) = \triangleright^{b,b}(\mathcal{P}_h) = c(a,b)$*.*
- $r = c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(x))$ *est exempt de* $c(a,b)$ *modulo* $[\![]\!]_e$*, puisque* x *ne peut être instancié que par* a *ou* b*, donc* $[\![u]\!]_e = [\![c(id(b), id(a))]\!]_e \cup [\![c(id(b), id(b))]\!]_e$*. De plus, comme* $\triangleright^{\mathcal{P}_{id}}(a) = b$ *et* $\triangleright^{\mathcal{P}_{id}}(b) = \bot$*, on peut conclure que* $[\![c(id(b), id(a))]\!]_e = \{c(a,a), c(b,a)\}$ *et* $[\![c(id(b), id(b))]\!]_e = \{c(a,a), c(b,b)\}$*.*

Cependant, la sémantique exacte fonctionne par évaluation récursive de toutes les instances closes du terme considéré pour pouvoir reconnaître les différentes occurrences d'un même sous-terme. Étant donné qu'il y a en pratique une potentielle infinité de ces instances, ce n'est généralement pas un formalisme facilement utilisable pour l'évaluation de propriété d'Exemption de Motif et l'analyse statique d'un CBTRS.

On a donc introduit la notion de sémantique substitutive qui adopte une approche par substitution et par évaluation récursive similaire à celle de la sémantique exacte. Cependant, la sémantique substitutive n'applique pas concrètement la substitution considérée et ne prend donc en compte les contraintes de corrélation entre deux sous-termes identiques que quand elles sont déjà apparentes dans le terme d'origine. Cette approche permet ainsi d'exprimer correctement toutes les contraintes de corrélation entre plusieurs instances d'une même variable, sans nécessairement demander l'évaluation complète du terme pour une potentielle infinité de substitutions.

Exemple 5.7. *On reprend l'algèbre de termes considérée dans l'Exemple 5.5. On considère des propriétés d'Exemption de Motif modulo* $[\![]\!]_s$ *:*

- $t = c(g(x), g(x))$ *est exempt de* $c(a,b)$ *modulo* $[\![]\!]_s$*, puisque la variable* x *ne peut être instanciée que par* a *ou* b*. On note* σ_a *et* σ_b *les substitutions respectives et on a donc* $[\![t]\!]_s = [\![t]\!]_{\sigma_a} \cup [\![t]\!]_{\sigma_b}$*. Comme* $\triangleright^{\mathcal{P}_g}(a) = \triangleright^{\mathcal{P}_g}(b) = \bot$*, on a donc* $[\![t]\!]_e = \{c(a,a), c(b,b)\}$*.*

- $u = h(x, x)$ *est exempt de* $c(a, b)$ *modulo* $[\![]\!]_s$*, puisque* x *ne peut être instancié que par* a *ou* b*. On note* σ_a *et* σ_b *les substitutions respectives et on a donc* $[\![u]\!]_s = [\![u]\!]_{\sigma_a} \cup [\![u]\!]_{\sigma_b}$*. Comme* $\triangleright^{\mathcal{P}_h}(a, a) = \triangleright^{\mathcal{P}_h}(b, b) = c(a, b)$*, on a donc* $[\![u]\!]_s = \{v \in \mathcal{T}_{\mathsf{S1}}(\mathcal{C}) \mid v \text{ exempt de } c(a, b)\}$*.*
- $r = c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(x))$ *n'est pas exempt de* $c(a, b)$ *modulo* $[\![]\!]_s$*, puisque même si* x *ne peut être instancié que par* a *ou* b*, en notant* σ_a *et* σ_b *les substitutions respectives, on a* $[\![r]\!]_s = [\![r]\!]_{\sigma_a} \cup [\![r]\!]_{\sigma_b}$*. Or comme* $\triangleright^{\mathcal{P}_{id}}(a) = b$ *et* $\triangleright^{\mathcal{P}_{id}}(a) = \bot$*, on a bien* $[\![r]\!]_{\sigma_a} = \{c(a, a), c(b, a)\}$ *mais* $[\![r]\!]_{\sigma_b} = \mathcal{T}_{\mathsf{S1}}(\mathcal{C})$ *puisque les deux sous-termes* $id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b)$ *et* $id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(x)$ *sont considérés indépendamment par la sémantique substitutive, malgré la substitution* σ_b *considérée. On a donc bien* $[\![u]\!]_s = \mathcal{T}_{\mathsf{S1}}(\mathcal{C})$*.*

Comme on l'a illustré avec l'étude de chacune de ces notions de sémantique, on peut les ordonner par précision :

Proposition 5.14. *Soit un terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$*, on a :*

$$[\![t]\!]_e \subseteq [\![t]\!]_s \subseteq [\![t]\!]_c \subseteq [\![t]\!]_g$$

Démonstration. Soit t un terme de $\mathcal{T}(\mathcal{F}, \mathcal{X}^a)$. On peut commencer la preuve de la Proposition en remarquant que si $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ on a clairement $[\![t]\!]_e = [\![t]\!]_s = [\![t]\!]_c = [\![t]\!]_g$.

On prouve en Annexe A le Lemme suivant :

Lemme 5.15. *Soient un terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$*, pour toute position* $\omega \in \mathcal{P}os(t)$ *telle que* $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ *avec* $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$*, on a* $[\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_c \subseteq [\![t]\!]_c$ *avec* z_s^{-p} *une variable fraîche et* $p = \triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$*.*

On prouve maintenant par induction sur k le nombre de symboles définis de t qu'on a $[\![t]\!]_s \subseteq [\![t]\!]_c \subseteq [\![t]\!]_g$. Pour $k = 0$, on a $t \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, donc l'inclusion est vérifiée. On considère maintenant $k > 0$ tel que pour tout terme u ayant strictement moins de k symboles définis, on a $[\![u]\!]_s \subseteq [\![u]\!]_c \subseteq [\![u]\!]_g$:

- Soit $w \in [\![t]\!]_s$, par définition, il existe une substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$ telle que $w \in [\![t]\!]_\varsigma$, *i.e.* il existe une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ où $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$ et pour tout $i \in [1, n]$ $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ telle que $w \in [\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_{\varsigma'}$ avec z_s^{-p} une variable fraîche où $p = \triangleright^{\mathcal{P}}(\varsigma(t_1), \ldots, \varsigma(t_n))$, et ς' une substitution valeur telle que $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$ et $\varsigma'(z_s^{-p}) \in \mathcal{T}_s(\mathcal{C})$ exempt de p. On a donc $w \in [\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_s$, d'où, avec l'hypothèse d'induction, $w \in [\![\varsigma'(t\left[t_{|\omega} \mapsto z_s^{-p}\right])]\!]_c$. Comme $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ pour tout $i \in [1, n]$, on a $p = \triangleright^{\mathcal{P}}(t_1, \ldots, t_n) = \triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$, donc, d'après le Lemme précédent, $w \in [\![t]\!]_c$.
- Soit $w \in [\![t]\!]_c$, par définition, il existe une position $\omega \in \mathcal{P}os(t)$ où $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$ telles que $w \in [\![t\left[t_{|\omega} \mapsto v\right]]\!]_c$. D'après l'hypothèse d'induction, on a donc $w \in [\![t\left[t_{|\omega} \mapsto v\right]]\!]_g$. Par définition de la sémantique close, on peut donc conclure que $w \in [\![t]\!]_g$.

Avec le même raisonnement, on a par induction sur k, que pour tout $t \in \mathcal{T}(\mathcal{F})$, on a $[\![t]\!]_e \subseteq [\![t]\!]_s$.

On considère maintenant $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ et $w \in [\![t]\!]_e$. Par définition, il existe une substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$ telle que $w \in [\![\varsigma(t)]\!]_e$. Comme $\varsigma(t) \in \mathcal{T}(\mathcal{F})$, on a donc $w \in [\![\varsigma(t)]\!]_e = [\![\varsigma(t)]\!]_s = [\![\varsigma(t)]\!]_\varsigma \subseteq [\![t]\!]_\varsigma \subseteq [\![t]\!]_s$. □

Bien que la sémantique exacte soit donc la plus précise, en pratique elle requiert d'être évaluée pour une potentielle infinité de substitutions. On verra, dans la Section suivante, que pour la sémantique substitutive, on peut donner une procédure de décision des propriétés d'Exemption de Motif associées. On se concentrera donc, par la suite, plus particulièrement sur l'étude de cette sémantique. L'adaptation de la méthode d'analyse à la notion de sémantique confluente a été présentée dans une publication [CLM21], mais, étant donné que cette dernière est moins précise, on lui préfèrera la sémantique substitutive. Pour permettre une analyse similaire, on propose, dans un premier temps, de transcrire les notions de préservation de sémantique dans le formalisme correspondant à la sémantique substitutive.

5.2.2 Préservation de sémantique dans un système non-linéaire

Comme présenté dans le Chapitre 3, chaque profil annotant un symbole exprime une hypothèse sur le comportement de la réduction associée à ce symbole. Il est donc nécessaire de vérifier que le CBTRS décrivant la réduction en question est bien en accord avec le choix d'annotation fait. En pratique, cela revient à prouver que la relation de réécriture induite par ce CBTRS préserve la sémantique considérée.

Dans le Chapitre 3, on avait ainsi introduit une notion de règle (et par extension de système de réécriture) préservant la sémantique $[\![]\!]_g$ (Définition 3.14). Cette dernière établit qu'une règle de réécriture $ls \rightharpoonup rs$ préserve la sémantique $[\![]\!]_g$ si et seulement si, pour toute substitution σ, on a $[\![\sigma(rs)]\!]_g \subseteq [\![\sigma(ls)]\!]_g$. On introduit donc une notion similaire de préservation de la sémantique substitutive :

Définition 5.8 (Préservation de sémantique substitutive)**.** *On dit qu'une règle de réécriture $ls \rightharpoonup rs$, préserve la sémantique substitutive $[\![]\!]_s$ si et seulement, pour toute substitution σ, on a pour toute substitution valeur ς telle que $\mathcal{V}ar(\sigma(ls)) \subseteq \mathcal{D}om(\varsigma)$, $[\![\sigma(rs)]\!]_\varsigma \subseteq [\![\sigma(ls)]\!]_\varsigma$.*

On dit qu'un TRS $\mathcal{R}$ préserve la sémantique substitutive $[\![]\!]_s$ si et seulement si toutes les règles de $\mathcal{R}$ préservent la sémantique substitutive.

La notion de sémantique substitutive reposant sur une approche de sur-approximation séparant instanciation des variables et évaluation des formes normales des sous-termes, sa préservation conserve cette logique, en distinguant donc la substitution considérée par la relation de réécriture de celle utilisée pour évaluer la sémantique substitutive.

Bien qu'un TRS préservant la sémantique $[\![]\!]_g$ induit une relation de réécriture qui préserve également la sémantique (Proposition 3.15), ce n'est pas forcément le cas pour la préservation de la sémantique substitutive. En effet, comme cette dernière considère simultanément une sur-approximation de tous les sous-termes identiques, elle n'est plus nécessairement préservée à chaque pas de réduction mais la préservation est garantie sur plusieurs étapes :

Proposition 5.16. *Étant donné un CBTRS $\mathcal{R}$ préservant la sémantique substitutive $[\![]\!]_s$, pour tous termes clos $t, u \in \mathcal{T}(\mathcal{F})$ tels que $t \Longrightarrow_\mathcal{R} u$, il existe un terme clos $v \in \mathcal{T}(\mathcal{F})$ tel que $u \Longrightarrow_\mathcal{R}^* v$ et $[\![v]\!]_s \subseteq [\![t]\!]_s$.*

Démonstration. On prouve en Annexe A le Lemme suivant :

Lemme 5.17. *Étant donnés une règle de réécriture $ls \rightharpoonup rs$, avec $ls = \varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n)$ où $\varphi \in \mathcal{D}^n$, préservant la sémantique substitutive $[\![]\!]_s$, un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et une substitution σ^t tels qu'il existe une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \sigma^t(ls)$, alors on a pour toute substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$:*

$$[\![t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]]\!]_\varsigma \subseteq [\![t]\!]_\varsigma$$

Soient un CBTRS $\mathcal{R}$ préservant la sémantique $[\![]\!]_s$ et des termes $t, u \in \mathcal{T}(\mathcal{F})$ tels que $t \Longrightarrow_{\mathcal{R}} u$. Par définition, il existe donc une règle $ls \twoheadrightarrow rs \in \mathcal{R}$, une position $\omega \in \mathcal{P}os(t)$ et une substitution σ^t telles que $t_{|\omega} = \sigma^t(ls)$ et $u = t\left[\sigma^t(rs)\right]_\omega$.

On note $\omega_1 := \omega$ et $\omega_2, \ldots, \omega_m$ les positions distinctes de t telles que $t_{|\omega_i} = \sigma^t(ls)$, en appliquant la règle $ls \twoheadrightarrow rs \in \mathcal{R}$ $m-1$ fois sur le terme u on a donc $u \Longrightarrow^*_{\mathcal{R}} v$ avec $v := t\left[t_{|\omega} \mapsto \sigma^t(rs)\right]$. De plus, avec le Lemme précédent, on a bien $[\![v]\!]_s \subseteq [\![t]\!]_s$. □

Et de même, la préservation des propriétés d'Exemption de Motif modulo $[\![]\!]_s$ n'est pas garantie à chaque pas de réduction mais ces dernières seront nécessaire récupérées après plusieurs étapes :

Corollaire 5.18. *Étant donnés un CBTRS $\mathcal{R}$ préservant la sémantique substitutive $[\![]\!]_s$ et un motif étendu p, pour tous termes clos $t, u \in \mathcal{T}(\mathcal{F})$ tels que $t \Longrightarrow_{\mathcal{R}} u$, si t est exempt de p modulo $[\![]\!]_s$, alors il existe un terme clos $v \in \mathcal{T}(\mathcal{F})$ exempt de p modulo $[\![]\!]_s$ tel que $u \Longrightarrow^*_{\mathcal{R}} v$.*

Démonstration. La preuve est immédiate par Proposition 5.16 et la définition de l'Exemption de Motif modulo sémantique. □

Dans le cadre d'une relation de réécriture confluente, la préservation de la sémantique substitutive d'un CBTRS non-linéaire, fournit donc une condition suffisante à la préservation (sur plusieurs étapes de réduction) de la sémantique $[\![]\!]_s$ et donc des propriétés d'Exemption de Motif modulo $[\![]\!]_s$. Comparé à la notion de préservation de la sémantique $[\![]\!]_g$ vérifiée par la méthode d'analyse présentée dans le Chapitre précédent, cette condition est d'autant plus faible que la sémantique substitutive est plus précise.

Exemple 5.8. *On reprend l'algèbre de termes considérée dans l'Exemple 3.7 et on considère le système $\mathcal{R}$ présenté dans l'Exemple 5.4 :*

$$\begin{cases} f(c(x,a)) & \twoheadrightarrow & c(x,x) \\ f(c(x,b)) & \twoheadrightarrow & h(x,x) \\ f(d(x)) & \twoheadrightarrow & c(g(x),g(x)) \\ g(c(x,y)) & \twoheadrightarrow & a \\ g(d(x)) & \twoheadrightarrow & b \\ h(x,y) & \twoheadrightarrow & c(y,x) \end{cases}$$

où les symboles définis $f^{!\mathcal{P}_f}, g^{!\mathcal{P}_g}$ et $h^{!\mathcal{P}_h}$ sont annotés avec $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}, \mathcal{P}_g = \emptyset$ et $\mathcal{P}_h = \{b * \bot \mapsto c(a,b), \bot * a \mapsto c(a,b)\}$.

On a vu dans l'Exemple 5.4, que l'on ne pouvait pas vérifier la préservation de la sémantique $[\![]\!]_g$ des deuxième et troisième règles. Or, comme on l'a montré dans l'Exemple 5.7, les termes $r_2 = h(x,x)$ et $r_3 = c(g(x),g(x))$ sont exempts de $c(a,b)$ modulo $[\![]\!]_s$. Donc pour toute substitution σ, on a bien $[\![\sigma(r_2)]\!]_s \subseteq [\![\sigma(f^{!\mathcal{P}_f}(c(x,b)))]\!]_s = [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$ et $[\![\sigma(r_3)]\!]_s \subseteq [\![\sigma(f^{!\mathcal{P}_f}(d(x))]\!]_s = [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.

Les autres règles préservaient déjà la sémantique $[\![]\!]_g$ et on peut vérifier trivialement qu'elles préservent donc toujours la sémantique substitutive $[\![]\!]_s$. Avec la Proposition 5.16, on peut alors conclure, que comme le système $\mathcal{R}$ est complet, confluent et terminant, pour tout terme $e \in \mathcal{T}_{\mathsf{S1}}(\mathcal{F})$, $f(e)$ sera réduit à une valeur exempte de $c(a,b)$.

La notion de sémantique exprimant toujours une sur-approximation des formes normales potentielles du terme considéré, comme pour la Proposition 3.15, la condition de préservation de la sémantique par une relation de réécriture explicitée par la Proposition 5.16 est suffisante mais non nécessaire. Et de façon similaire, plus les annotations seront précises, moins la sur-approximation sera grossière, réduisant ainsi les possibilités d'un faux négatif.

Exemple 5.9. *On reprend l'algèbre de termes considérée dans l'Exemple 3.7 et le système $\mathcal{R}$ présenté dans l'Exemple 5.4 :*

$$\begin{cases} f(c(x,y)) & \rightarrowtriangle & c(x,x) \\ f(d(x)) & \rightarrowtriangle & c(id(b),id(x)) \\ g(c(x,y)) & \rightarrowtriangle & a \\ g(d(x)) & \rightarrowtriangle & b \\ id(x) & \rightarrowtriangle & x \end{cases}$$

où les symboles définis $f^{!\mathcal{P}_f}, g^{!\mathcal{P}_g}$ et $id^{!\mathcal{P}_{id}}$ sont annotés avec $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}, \mathcal{P}_g = \emptyset$ et $\mathcal{P}_{id} = \{b \mapsto b\}$.

Comme on l'a montré dans l'Exemple 5.7, le terme $c(id(b),id(x))$ n'est pas exempt de $c(a,b)$ modulo $[\![]\!]_s$. On peut en déduire que la deuxième règle de $\mathcal{R}$ ne préserve donc pas la sémantique $[\![]\!]_s$. Cependant, en ajoutant le profil $a \mapsto a$ à l'annotation du symbole $id^{!\mathcal{P}_{id}}$, ce même terme est ainsi bien exempt de $c(a,b)$ et on voit alors que le système $\mathcal{R}$ préserve la sémantique.

En pratique, il est plus simple de considérer pour chaque règle, qu'elle respecte les propriétés d'Exemption de Motif induites par chaque profil :

Définition 5.9 (Satisfaction substitutive de profil)**.** *Soit une règle de réécriture $ls \rightarrowtriangle rs$ avec $ls = \varphi_s^{!\mathcal{P}}(ls_1,\dots,ls_n)$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$, on dit que la règle satisfait le profil π par substitution si et seulement pour toute substitution valeur ς avec $\mathcal{V}ar(ls) \subseteq \mathcal{D}om(\varsigma)$, on a :*

$$\varsigma(ls_i) \text{ est exempt de } p_i, \text{ pour tout } i \in [1,n] \implies rs \text{ est exempt de } p \text{ modulo } [\![]\!]_\varsigma$$

La satisfaction de tous les profils annotant le symbole défini en tête du membre gauche d'une règle permet en effet de vérifier la préservation de la sémantique :

Proposition 5.19. *Soit une règle de réécriture $\varphi_s^{!\mathcal{P}}(ls_1,\dots,ls_n) \rightarrowtriangle rs$, la règle préserve la sémantique substitutive si et seulement elle satisfait par substitution tous les profils de $\mathcal{P}$.*

Démonstration. Soient une règle de réécriture $ls \rightarrowtriangle rs$ avec $ls = \varphi_s^{!\mathcal{P}}(ls_1,\dots,ls_n)$ satisfaisant par substitution tous les profils de $\mathcal{P}$, et une substitution σ. On prouve par induction sur k le nombre de symboles définis dans $\sigma(ls)$ qu'on a pour toute substitution valeur ς avec $\mathcal{V}ar(\sigma(ls)) \subseteq \mathcal{D}om(\varsigma)$, $[\![\sigma(rs)]\!]_\varsigma \subseteq [\![\sigma(ls)]\!]_\varsigma$.

- Si $k = 1$, alors φ est le seul symbole défini de $\sigma(ls)$, et $\varsigma \circ \sigma$ est donc une substitution valeur telle que $\mathcal{V}ar(ls) \subseteq \mathcal{D}om(\varsigma \circ \sigma)$. Par définition, on a $[\![\sigma(ls)]\!]_\varsigma = [\![x_s^{-p}]\!]$ avec $p := \triangleright^{\mathcal{P}}(\varsigma(\sigma(ls_1)),\dots,\varsigma(\sigma(ls_n)))$, et d'après le Lemme 5.12 on a $[\![\sigma(rs)]\!]_\varsigma = [\![\sigma(rs)]\!]_{\varsigma\circ\sigma} \subseteq [\![rs]\!]_{\varsigma\circ\sigma}$. Or, comme la règle satisfait par substitution tous les profils de $\mathcal{P}$, pour tout profil $p_1 * \cdots * p_n \mapsto q \in \mathcal{P}$ tel que $\varsigma(\sigma(ls_i))$ est exempt de p_i pour tout $i \in [1,n]$, rs est exempt de q modulo $[\![]\!]_{\varsigma\circ\sigma}$. Donc, par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, rs est exempt de $p = \triangleright^{\mathcal{P}}(\varsigma(\sigma(ls_1)),\dots,\varsigma(\sigma(ls_n)))$ modulo $[\![]\!]_{\varsigma\circ\sigma}$. Par définition de l'Exemption de Motif modulo sémantique, on a donc bien $[\![\sigma(rs)]\!]_\varsigma \subseteq [\![x_s^{-p}]\!] = [\![\sigma(ls)]\!]_\varsigma$.

- On considère maintenant $k > 0$, tel que pour toute substitution σ' tel que $\sigma'(ls)$ a strictement moins de k symboles définis, on a pour toute substitution valeur ς avec $\mathcal{V}ar(\sigma'(ls)) \subseteq \mathcal{D}om(\varsigma)$ $[\![\sigma'(rs)]\!]_\varsigma \subseteq [\![\sigma'(ls)]\!]_\varsigma$. Comme $k > 1$, il existe une position $\omega \in \mathcal{P}os(\sigma(ls))$ telle que $\sigma(ls)_{|\omega} = \psi_{s'}^{!\mathcal{P}'}(t_1,\dots,t_m)$ avec $\psi \in \mathcal{D}^m$ et $t_i \in \mathcal{T}(\mathcal{C},\mathcal{X})$ pour tout i, et pour toute substitution valeur ς telle que $\mathcal{V}ar(\sigma(ls)) \subseteq \mathcal{D}om(\varsigma)$, on a donc $[\![\sigma(ls)]\!]_\varsigma = [\![\sigma(ls)\left[\sigma(ls)_{|\omega} \mapsto z_{s'}^{-p}\right]]\!]_{\varsigma'}$

avec $z_{s'}^{-p}$ une variable fraîche et ς' une substitution valeur telle que $\varsigma'(z_{s'}^{-p})$ est exempt de $p := \triangleright^{\mathcal{P}'}(\varsigma(t_1), \ldots, \varsigma(t_m))$ et $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$.

On note $\sigma' := \{x \mapsto \sigma(x)\left[\sigma(ls)_{|\omega} \mapsto z_{s'}^{-p}\right]\}$, et par construction on a alors $\sigma'(ls) = \sigma(ls)\left[\sigma(ls)_{|\omega} \mapsto z_{s'}^{-p}\right]$. De plus, en notant $\sigma^z = \{z_{s'}^{-p} \mapsto \sigma(ls)\}$, on a $\sigma^z(\sigma'(rs)) = \sigma(rs)$, donc d'après le Lemme 5.12, on a $[\![\sigma(rs)]\!]_\varsigma \subseteq \bigcup_{\varsigma'} [\![\sigma'(rs)]\!]_{\varsigma'}$ avec ς' une substitution valeur telle que $\varsigma'(z_{s'}^{-p}) \in [\![\sigma(ls)_{|\omega}]\!]_\varsigma = [\![x_{s'}^{-p}]\!]$ et $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$.

Enfin, par hypothèse d'induction, on a $[\![\sigma'(rs)]\!]_{\varsigma'} \subseteq [\![\sigma'(ls)]\!]_{\varsigma'}$, et on peut donc conclure que $[\![\sigma(rs)]\!]_\varsigma \subseteq [\![\sigma(ls)]\!]_\varsigma$.

Par induction, pour toute substitution σ et pour toute substitution valeur ς avec $\mathcal{V}ar(\sigma(ls)) \subseteq \mathcal{D}om(\varsigma)$, on a donc $[\![\sigma(rs)]\!]_\varsigma \subseteq [\![\sigma(ls)]\!]_\varsigma$, *i.e.* la règle $ls \twoheadrightarrow rs$ préserve la sémantique $[\![]\!]_s$.

On suppose maintenant qu'il existe un profil $p_1 * \cdots * p_n \mapsto q$ non satisfait par la règle $ls \twoheadrightarrow rs$. Par définition, il existe donc une substitution valeur ς avec $\mathcal{V}ar(ls) \subseteq \mathcal{D}om(\varsigma)$ telle que $\varsigma(ls_i)$ est exempt de p_i pour tout $i \in [1, n]$ et rs n'est pas exempt de q modulo $[\![]\!]_\varsigma$. Par définition de la sémantique modulo sémantique $[\![ls]\!]_\varsigma = [\![x_s^{-p}]\!] \subseteq [\![x_s^{-q}]\!]$ avec $p := \triangleright^{\mathcal{P}}(\varsigma(ls_1), \ldots, \varsigma(ls_n))$. Comme rs n'est pas exempt de q modulo $[\![]\!]_\varsigma$, il existe $v \in [\![rs]\!]_\varsigma$ tel que $v \notin [\![x_s^{-p}]\!]$. Donc $[\![rs]\!]_\varsigma \not\subseteq [\![ls]\!]_\varsigma$, *i.e.* la règle $ls \twoheadrightarrow rs$ ne préserve pas la sémantique $[\![]\!]_s$. □

Comme présenté dans le Chapitre 3, l'annotation d'un symbole défini exprime une hypothèse quant au comportement de la fonction associée au symbole : pour chaque profil on suppose qu'un terme vérifiant la pré-condition exprimée par le membre gauche du profil devra être réduit en un terme vérifiant la post-condition exprimée par son membre droit. La méthode d'analyse présentée dans le Chapitre 4 proposait ainsi de vérifier la correction de ces hypothèses dans le cas de systèmes linéaires. Étant donné le formalisme présenté ici pour l'étude de systèmes non-linéaires, on présente dans la Section suivante une méthode basée sur ce formalisme et adaptée à l'analyse de systèmes non-linéaires.

5.3 Méthode d'analyse non-linéaire

Comme dans le Chapitre précédent, on propose donc une méthode d'analyse statique en se reposant sur la notion de satisfaction substitutive de profil. Une telle méthode doit donc pouvoir être appliquée de façon systématique, sans nécessiter l'instanciation des membres gauches et droits de chaque règle via une potentielle infinité de substitutions valeurs.

En effet, comme pour la notion de satisfaction d'origine (Définition 3.15), la satisfaction par substitution dit qu'une règle satisfait un de ses profils si et seulement respecter la pré-condition exprimée par le membre gauche du profil implique de respecter la post-condition décrite par son membre droit. Comme précédemment, on se reposera donc sur le mécanisme d'aliasing pour inférer les contraintes d'instanciations des variables.

Cette méthode d'analyse fonctionne ainsi de façon très similaire à celle proposée dans le Chapitre précédent. Cependant, dans le cas de la sémantique substitutive, prendre en compte les contraintes de corrélation, entre les différentes instances d'une même variable, rend difficile la construction d'un motif constructeur servant d'équivalent sémantique aux termes considérés. On propose donc de s'abstenir de l'étape de construction de l'équivalent sémantique et on proposera donc un mécanisme de décision permettant de conclure quant à la satisfaction sans utiliser d'équivalent sémantique. La méthode se présente donc en deux étapes :

- Une étape d'inférence, basée sur le mécanisme d'aliasing, des substitutions considérées par la notion de satisfaction de profil ;
- Une étape de calcul des sémantiques permettant de vérifier les propriétés d'Exemption de Motif.

On présente dans un premier temps comment le mécanisme d'aliasing permet l'inférence pour un système non-linéaire.

5.3.1 Inférence dans les systèmes non-linéaires

Pour pouvoir vérifier qu'une règle de réécriture $f^{|\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs$ satisfait par substitution un profil $p_1 * \ldots * p_n \mapsto p \in \mathcal{P}$, on veut caractériser les substitutions ς considérées en construisant des versions aliassées des motifs ls_i. En effet, on cherche à inférer les contraintes d'instanciations telles qu'on ait $\varsigma(ls_i)$ exempt de p_i, pour tout $i \in [1, n]$. On peut donc s'inspirer de l'approche présentée dans la Section 4.3 pour encoder l'ensemble de ces substitutions, en remarquant que, comme les substitutions ς considérées sont des substitutions valeurs, on a $\varsigma(ls_i) \in [\![ls_i]\!] \cap [\![x_{s_i}^{-p_i}]\!] = [\![ls_i \times x_{s_i}^{-p_i}]\!]$ (avec $x_{s_i}^{-p_i}$ une variable fraîche).

La notion de version aliassée des motifs ls_i permet en effet de donner une caractérisation des substitutions valeurs en inférant les contraintes sur les variables du motif :

Lemme 5.20. *Soit λ une version aliassée d'un terme $l \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, pour toute substitution valeur ς telle que $\varsigma(l) \in \mathcal{T}(\mathcal{C})$, on a $\varsigma(l) \in [\![\lambda]\!]$ si et seulement si $\varsigma(x) \in [\![\lambda_{@x}]\!]$, pour toute variable $x \in \mathcal{V}ar(l)$.*

Démonstration. Soit λ une version aliassée d'un terme $l \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, et une substitution valeur ς telle que $\varsigma(l) \in \mathcal{T}(\mathcal{C})$. Par définition, $\varsigma(l) \in [\![\lambda]\!]$ si et seulement si il existe une substitution σ telle que $\lambda \overset{\sigma}{\lessdot} \varsigma(l)$. On prouve alors par induction sur la forme de l que pour toute substitution σ on a $\lambda \overset{\sigma}{\lessdot} \varsigma(l)$ si et seulement si, pour tout $x \in \mathcal{V}ar(l)$, $\lambda_{@x} \overset{\sigma}{\lessdot} \varsigma(x)$.

Si $l = x_s \in \mathcal{X}$, alors $\lambda = x \mathbin{@} t$ et on a donc $\lambda_{@x} = t$, pour toute substitution σ, $\lambda \overset{\sigma}{\lessdot} \varsigma(l)$ si et seulement si $t \overset{\sigma}{\lessdot} \varsigma(x)$.

Si $l = c(l_1, \ldots, l_n)$ avec $c \in \mathcal{C}^n$, alors $\lambda = c(\lambda_1, \ldots, \lambda_n)$ avec λ_i une version aliasée de l_i pour tout $i \in [1, n]$. Par induction, on a pour tout $i \in [1, n]$, pour toute substitution σ, $\lambda_i \overset{\sigma}{\lessdot} \varsigma(l_i)$ si et seulement si, pour tout $x \in \mathcal{V}ar(l_i)$, $\lambda_{i@x} \overset{\sigma}{\lessdot} \varsigma(x)$. On prouve les deux implications séparément :

- On considère une substitution σ telle que $\lambda \overset{\sigma}{\lessdot} \varsigma(l)$. Par définition, pour tout $i \in [1, n]$, $\lambda_i \overset{\sigma}{\lessdot} \varsigma(l_i)$. Donc, par induction, on a, pour tout $i \in [1, n]$, pour toute variable $x \in \mathcal{V}ar(l_i)$, $\lambda_{i@x} \overset{\sigma}{\lessdot} \varsigma(x)$. $\varsigma(x) \in [\![\lambda_{i@x}]\!]$. De plus, pour tout $x \in \mathcal{V}ar(l)$, par construction on a $\lambda_{@x} = \Pi_i \lambda_{i@x}$ avec $i \in [1, n]$ tel que $x \in \mathcal{V}ar(l_i)$. On a donc, pour tout $x \in \mathcal{V}ar(l)$, $\lambda_{@x} \overset{\sigma}{\lessdot} \varsigma(x)$.
- On considère σ telle que, pour tout $x \in \mathcal{V}ar(l)$, $\lambda_{i@x} \overset{\sigma}{\lessdot} \varsigma(x)$. $\varsigma(x) \in [\![\lambda_{@x}]\!]$. De plus, pour tout $x \in \mathcal{V}ar(l)$, par construction on a $\lambda_{@x} = \Pi_i \lambda_{i@x}$ avec $i \in [1, n]$ tel que $x \in \mathcal{V}ar(l_i)$. Donc, pour tout $i \in [1, n]$, pour tout $x \in \mathcal{V}ar(l_i)$, $\lambda_{i@x} \overset{\sigma}{\lessdot} \varsigma(x)$. Par induction, on a pour tout $i \in [1, n]$, $\lambda_i \overset{\sigma}{\lessdot} \varsigma(l_i)$, donc $\lambda \overset{\sigma}{\lessdot} \varsigma(l)$.

Si $\varsigma(l) \in [\![\lambda]\!]$, il existe donc une substitution σ telle que, pour tout $x \in \mathcal{V}ar(l)$, $\lambda_{@x} \overset{\sigma}{\lessdot} \varsigma(x)$, d'où $\varsigma(x) \in [\![\lambda_{@x}]\!]$.

On suppose maintenant que pour tout $x \in \mathcal{V}ar(l)$, $\varsigma(x) \in [\![\lambda_{@x}]\!]$, *i.e.* il existe une substitution σ_x telle que $\lambda_{@x} \overset{\sigma_x}{\lessdot} \varsigma(x)$. Par définition de la version aliassée λ de l, on a pour toute variables

$x, y \in \mathcal{V}ar(l)$ distinctes, $\mathcal{MV}(\lambda_{@x}) \cap \mathcal{MV}(\lambda_{@y}) = \emptyset$. Donc, en notant $\sigma := \{y \mapsto \sigma_x(y) \mid x \in \mathcal{V}ar(l), y \in \mathcal{MV}(\lambda_{@x})\}$, on a, pour tout $x \in \mathcal{V}ar(l)$, $\lambda_{@x} \overset{\sigma}{\preccurlyeq} \varsigma(x)$. Par conséquent, on a $\lambda \overset{\sigma}{\preccurlyeq} \varsigma(l)$, et on peut donc conclure que $\varsigma(l) \in [\![\lambda]\!]$. □

En utilisant ce Lemme, on peut donc inférer les contraintes induites par la pré-condition de la satisfaction substitutive de profil sur les variables de la règle en utilisant le système $\mathfrak{R}$, présenté en Figure 4.8, pour exprimer les conjonctions $ls_i \times x_{s_i}^{-p_i}$ comme une somme de versions aliassées de ls_i (Proposition 4.13).

Exemple 5.10. *On reprend l'algèbre de termes définie par la signature Σ_{nl} considérée dans l'Exemple 3.7 et on étudie la règle de réécriture suivante*

$$\varphi(c(x,y),x) \quad \twoheadrightarrow \quad c(y,y)$$

où le symbole défini $\varphi^{!\mathcal{P}} : \mathsf{S1} * \mathsf{S2} \mapsto \mathsf{S1}$ est annoté avec $\mathcal{P} = \{c(a,b) * b \mapsto b\}$.

Pour vérifier que cette règle préserve la sémantique substitutive, il faut vérifier qu'elle satisfait par substitution le profil $c(a,b) * b \mapsto b$. On infère donc les contraintes d'instanciation des substitutions valeurs ς respectant la pré-condition du profil, i.e. telle que $\varsigma(c(x,y))$ est une valeur exempte de $c(a,b)$ et $\varsigma(x)$ est une valeur exempte de b. En utilisant le système $\mathfrak{R}$, on obtient :

$$\begin{aligned}(\!|c(x,y) \times z_{1\mathsf{S1}}^{-c(a,b)}, x \times z_{2\mathsf{S2}}^{-b}|\!) \downarrow_{\mathfrak{R}} \quad = \quad & (\!|c(x @ (z_{3\mathsf{S2}}^{-c(a,b)} \setminus a), y @ z_{4\mathsf{S2}}^{-c(a,b)}), x @ z_{2\mathsf{S2}}^{-b}|\!) \\ & + (\!|c(x @ z_{3\mathsf{S2}}^{-c(a,b)}, y @ (z_{4\mathsf{S2}}^{-c(a,b)} \setminus b)), x @ z_{2\mathsf{S2}}^{-b}|\!)\end{aligned}$$

On a donc $[\![(\!|c(x,y) \times z_{1\mathsf{S1}}^{-c(a,b)}, x \times z_{2\mathsf{S2}}^{-b}|\!)]\!] = [\![\lambda^1]\!] \cup [\![\lambda^2]\!]$ avec $\lambda^1 = (\!|c(x @ (z_{3\mathsf{S2}}^{-c(a,b)} \setminus a), y @ z_{4\mathsf{S2}}^{-c(a,b)}), x @ z_{2\mathsf{S2}}^{-b}|\!)$ et $\lambda^2 = (\!|c(x @ z_{3\mathsf{S2}}^{-c(a,b)}, y @ (z_{4\mathsf{S2}}^{-c(a,b)} \setminus b)), x @ z_{2\mathsf{S2}}^{-b}|\!)$. Donc, pour toute substitution valeur ς telle que $\varsigma(c(x,y))$ est une valeur exempte de $c(a,b)$ et $\varsigma(x)$ est une valeur exempte de b, $\varsigma((\!|c(x,y),x|\!)) \in [\![\lambda^1]\!]$ ou $\varsigma((\!|c(x,y),x|\!)) \in [\![\lambda^2]\!]$. De plus, on peut vérifier que $\lambda^1_{@x} = (z_{3\mathsf{S2}}^{-c(a,b)} \setminus a) \times z_{2\mathsf{S2}}^{-b}$ est réduit à $\perp$ par $\mathfrak{R}$, donc, d'après le Lemme 5.6, $[\![\lambda^1]\!] = \emptyset$. On peut donc conclure, avec le Lemme 5.20, que $\varsigma(x) \in [\![z_{2\mathsf{S2}}^{-b}]\!]$ et $\varsigma(y) \in [\![z_{4\mathsf{S2}}^{-c(a,b)} \setminus b)]\!]$, ce qui permet d'encoder les contraintes d'instanciation par la substitution $\sigma^{@\lambda^2}_{(\!|c(x,y),x|\!)}$.

Les contraintes d'instanciation des variables du membre gauche sont ainsi inférées sous la forme d'une substitution $\sigma^{@\lambda}_{(\!|ls_1,\ldots,ls_n|\!)}$ aliassant les variables avec un motif quasi-symbolique. Appliquer ces contraintes au membre droit rs de la règle donne alors une versions aliassée $\rho = \sigma^{@\lambda}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)$ de rs. Afin d'exprimer la post-condition à vérifier sur le terme ainsi inféré en utilisant la sémantique, on étend la notion de sémantique substitutive pour de tels motifs :

Définition 5.10 (Sémantique substitutive étendue)**.** *Soit τ une version aliassée d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on définit la sémantique substitutive de τ par :*

$$[\![\tau]\!]_s = \bigcup_{\varsigma} [\![t]\!]_\varsigma$$

avec ς une substitution valeur telle que $\varsigma(x) \in [\![\tau_{@x}]\!]$, pour toute variable $x \in \mathcal{V}ar(t)$.

Grâce à la Proposition 4.13, on peut alors vérifier la satisfaction de profil en calculant, via le système $\mathfrak{R}$, l'intersection de la sémantique profonde de la forme inférée du membre droit de la règle considérée avec la sémantique close du membre droit du profil considéré :

Proposition 5.21. *Soit une règle $ls \twoheadrightarrow rs$ avec $ls = f_s^{!\mathcal{P}}(ls_1, \ldots, ls_n)$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$. On note $v := (\!|ls_1 \times z_{1s_1}^{-p_1}, \ldots, ls_n \times z_{ns_n}^{-p_n}|\!) \downarrow_{\mathfrak{R}}$, et on a :*

- *si* $v = \bot$ *alors la règle satisfait par substitution le profil* π *;*
- *sinon* $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ *avec chaque* $\lambda_i^j, j \in [1, m]$ *une version aliassée de* $ls_i, i \in [1, n]$*, la règle satisfait par substitution le profil* π *si et seulement si, pour tout* k
  - *soit il existe* $x \in \mathcal{V}ar(ls)$ *telle que* $(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)_{@x} \downarrow_{\mathfrak{R}} = \bot$ *;*
  - *soit* $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$.

Démonstration. Soit une règle $ls \twoheadrightarrow rs$ avec $ls = f_s^{!\mathcal{P}}(ls_1, \ldots, ls_n)$, et un profil $\pi = p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$. On note $v := (\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!) \downarrow_{\mathfrak{R}}$.

Si $v = \emptyset$, comme la Proposition 5.4 garantit que le système $\mathfrak{R}$ préserve la sémantique, il existe $k \in [1, n]$ tel que $[\![ls_k \times z_k s_k^{-p_k}]\!] = \emptyset$. Donc, pour toute substitution valeur ς, $\varsigma(ls_k)$ n'est pas exempt de p_k. Par conséquent, la règle satisfait par substitution le profil π.

On considère maintenant que $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$

- On suppose que pour tout $k \in [1, m]$ soit il existe $x \in \mathcal{V}ar(ls)$ telle que $(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)_{@x} \downarrow_{\mathfrak{R}} = \bot$, soit $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$. On considère une substitution valeur ς telle que $\varsigma(ls_i)$ est exempt de p_i, pour tout $i \in [1, n]$. D'après la Proposition 5.4, le système $\mathfrak{R}$ préserve la sémantique, donc il existe $k \in [1, m]$ tel que pour tout $i \in [1, n]$, $\varsigma(ls_i) \in [\![\lambda_i^k]\!]$. D'après le Lemme 5.20, on a donc $\varsigma(x) \in [\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(x)]\!] \neq \emptyset$ pour tout $x \in \mathcal{V}ar(ls)$. Par préservation de la sémantique, on a donc $(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)_{@x} \downarrow_{\mathfrak{R}} \neq \bot$ pour tout $x \in \mathcal{V}ar(ls)$, et par définition de la sémantique substitutive étendue, on a alors $[\![rs]\!]_\varsigma \subseteq [\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]$. Par conséquent, $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$, et, par définition, rs est donc exempt de p modulo $[\![]\!]_\varsigma$.
- On suppose que la règle satisfait par substitution le profil, et on considère $k \in [1, m]$. Par définition de la sémantique substitutive étendue, on a $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s = \bigcup_\varsigma [\![rs]\!]_\varsigma$ avec ς une substitution valeur telle que $\varsigma(x) \in [\![(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)_{@x}]\!]$. D'après le Lemme 5.20, on a donc $\varsigma(ls_i) \in [\![\lambda_i^k]\!]$ pour tout $i \in [1, n]$. De plus, d'après la Proposition 5.4, le système $\mathfrak{R}$ préserve la sémantique, donc pour tout $i \in [1, n]$, on a $[\![\lambda_i^k]\!] \subseteq [\![x_{s_i}^{-p_i}]\!]$, d'où $\varsigma(ls_i)$ exempt de p_i. Par satisfaction de profil, on a alors rs exempt de p modulo $[\![]\!]_\varsigma$, *i.e.* $[\![rs]\!]_\varsigma \subseteq [\![x_s^{-p}]\!]$. On a donc bien $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$.

□

Exemple 5.11. *On reprend l'algèbre de termes définie par la signature* Σ_{nl} *et la règle de réécriture suivante considérées dans l'Exemple 5.10*

$$\varphi(c(x, y), x) \quad \twoheadrightarrow \quad c(y, y)$$

où le symbole défini $\varphi^{!\mathcal{P}} : \mathsf{S1} * \mathsf{S2} \mapsto \mathsf{S1}$ *est annoté avec* $\mathcal{P} = \{c(a, b) * b \mapsto b\}$.

On avait montré dans l'Exemple précédent, qu'on a

$$(\!|c(x, y) \times z_1 \mathsf{S1}^{-c(a,b)}, x \times z_2 \mathsf{S2}^{-b}|\!) \downarrow_{\mathfrak{R}} = \lambda^1 + \lambda^2$$

avec $\lambda^1 = (\!|c(x \mathbin{@} (z_3 \mathsf{S2}^{-c(a,b)} \setminus a), y \mathbin{@} z_4 \mathsf{S2}^{-c(a,b)}), x \mathbin{@} z_2 \mathsf{S2}^{-b}|\!)$ *et* $\lambda^2 = (\!|c(x \mathbin{@} z_3 \mathsf{S2}^{-c(a,b)}, y \mathbin{@} (z_4 \mathsf{S2}^{-c(a,b)} \setminus b)), x \mathbin{@} z_2 \mathsf{S2}^{-b}|\!)$.

Comme $[\![\lambda^1]\!] = \emptyset$*, d'après la Proposition 5.21, la règle satisfait par substitution le profil* $c(a, b) * b \mapsto b$ *si et seulement* $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-b}]\!]$ *avec* $\rho = \sigma^{@\lambda^2}_{(\!|c(x,y),x|\!)}(rs)$*. On a donc* $\rho = c(y \mathbin{@} (z_4 \mathsf{S2}^{-c(a,b)} \setminus b), y \mathbin{@} (z_4 \mathsf{S2}^{-c(a,b)} \setminus b))$*, d'où* $[\![\rho]\!]_s = \{c(a, a)\} \subseteq [\![x_{\mathsf{S1}}^{-b}]\!]$*. On peut donc conclure que la règle satisfait par substitution le profil.*

Dans le cas général, comme le membre droit rs de la règle considérée peut contenir des symboles définis, vérifier qu'on a la post-condition $[\![\rho]\!]_s \subseteq [\![x_s^{-p}]\!]$, avec ρ une version aliassée de rs obtenue par inférence, est un problème assez compliqué. On proposera par la suite une procédure de décision permettant d'affirmer ou d'infirmer une telle relation.

5.3.2 Évaluation de sémantique par contexte

Étant donnés une règle de réécriture $ls \twoheadrightarrow rs$ et un profil $p_1 * \cdots * p_n \mapsto p$, on a montré, grâce à la Proposition 5.21, que pour établir que la règle satisfait par substitution le profil, il faut vérifier que pour une certaine version aliassée ρ de rs, on a $[\![\rho]\!]_s \subseteq [\![x_s^{-p}]\!]$, *i.e.* que $[\![\rho]\!]_s \setminus [\![x_s^{-p}]\!] = \emptyset$. On cherche donc à montrer que pour toute valeur $v \in [\![\rho]\!]_s$, on a $v \in [\![x_s^{-p}]\!]$, *i.e.* v est exempte de p. Si au contraire, on peut expliciter une valeur $v \in [\![\tau]\!]_s$ telle que v n'est pas exempte de p, la satisfaction du profil sera réfutée.

Afin d'expliciter l'existence d'une telle valeur, on introduit la notion de contexte d'évaluation qui permet de simuler l'évaluation de la sémantique substitutive d'un tel terme pour en construire les valeurs :

Définition 5.11 (Contexte d'évaluation)**.** *Étant donné un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on dit qu'un ensemble d'associations Γ est un* contexte d'évaluation *de t si et seulement si :*

- *pour toute variable $x \in \mathcal{V}ar(t)$, il existe une unique association $x \mapsto p \in \Gamma$, avec p un motif.*
- *pour toute position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi \in \mathcal{D}^n$, il existe une unique association $t_{|\omega} \mapsto p \in \Gamma$, avec p un motif.*

De plus, on dit que Γ est un contexte d'évaluation valeur si et seulement si, pour toute association $u \mapsto p \in \Gamma$, p est une valeur.

On peut alors évaluer la sémantique substitutive d'un terme en fonction d'un contexte donné :

Définition 5.12. *Étant donnés un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, et Γ un contexte d'évaluation de t, on appelle évaluation de la sémantique substitutive t suivant Γ, noté $[\![t]\!]_s^\Gamma$, l'ensemble défini par :*

- *si Γ est un contexte d'évaluation valeur, $[\![t]\!]_s^\Gamma$ est le singleton défini par :*
  - *$[\![c(t_1, \ldots, t_n)]\!]_s^\Gamma = \{c(v_1, \ldots, v_n)\}$ pour tout $c \in \mathcal{C}$, avec $[\![t_i]\!]_s^\Gamma = \{v_i\}$, pour tout $i \in [1, n]$;*
  - *$[\![\varphi(t_1, \ldots, t_n)]\!]_s^\Gamma = \{v\}$, pour tout $\varphi \in \mathcal{D}$, avec $\varphi(t_1, \ldots, t_n) \mapsto v \in \Gamma$;*
  - *$[\![x]\!]_s^\Gamma = \{v\}$, pour tout $x \in \mathcal{V}ar(t)$, avec $x \mapsto v \in \Gamma$;*
- *sinon, $[\![t]\!]_s^\Gamma = \bigcup_{\Gamma'} [\![t]\!]_s^{\Gamma'}$ avec Γ' un contexte d'évaluation valeur tel que pour toute association $u \mapsto p \in \Gamma$, il existe $v \in [\![p]\!]$ tel que $u \mapsto v \in \Gamma'$.*

Étant donnée τ une version aliassée de t, on dit que τ est évaluable par Γ si et seulement pour toute association $u \mapsto r \in \Gamma$:

- *si $u = x \in \mathcal{V}ar(t)$, alors $[\![r]\!] \subseteq [\![\tau_{@x}]\!]$;*
- *si $u = \varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n)$ avec $\varphi \in \mathcal{D}^n$, alors, pour tout $\Pi \subseteq \mathcal{P}$ tel qu'il existe $(\!|v_1, \ldots, v_n|\!) \in [\![(\!|u_1, \ldots, u_n|\!)]\!]_s^\Gamma$ avec $\Pi = \{p_1 * \cdots * p_n \mapsto p \in \mathcal{P} \mid \forall i \in [1, n], v_i$ est exempt de $p_i\}$, on a $[\![r]\!] \subseteq [\![x_s^{-\Pi^r}]\!]$ avec $\Pi^r = \sum_{p_1 * \cdots * p_n \mapsto q \in \Pi} q$.*

Pour une version aliassée d'un terme de l'algèbre, chaque valeur de sa sémantique peut être retrouvée par évaluation suivant un contexte :

Proposition 5.22. *Étant donnée une version aliassée τ d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $v \in [\![\tau]\!]_s$ si et seulement si il existe Γ, un contexte d'évaluation de t, tel que τ est évaluable par Γ et $v \in [\![t]\!]_s^\Gamma$.*

Démonstration. Soit τ une version aliassée d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on prouve par induction sur k le nombre de symboles définis dans t que pour toute substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$, on a $v \in [\![t]\!]_\varsigma$ si et seulement il existe un contexte d'évaluation valeur Γ tel que $v \in [\![t]\!]_s^\Gamma$ avec :

- $x \mapsto \varsigma(x) \in \Gamma$;
- pour toute association $\varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n) \mapsto w$ avec $\varphi \in \mathcal{D}^n$, w est exempt de $\triangleright^{\mathcal{P}}(w_1, \ldots, w_n)$ où $[\![(\!|u_1, \ldots, u_n|\!)]\!]_s^\Gamma = \{(\!|w_1, \ldots, w_n|\!)\}$.

Si $k = 0$, on a $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ et on montre par induction sur la forme de t que $[\![t]\!]_\varsigma = \{\varsigma(t)\} = [\![t]\!]_s^\Gamma$:

- si $t = x \in \mathcal{X}$, pour toute substitution valeur ς et pour tout contexte d'évaluation Γ avec $x \mapsto \varsigma(x) \in \Gamma$, on a donc $[\![t]\!]_\varsigma = \{\varsigma(x)\} = [\![t]\!]_s^\Gamma$.
- si $t = c(t_1, \ldots, t_n)$, pour toute substitution valeur ς et pour tout contexte d'évaluation Γ avec $x \mapsto \varsigma(x) \in \Gamma$, on a, par induction, $[\![t_i]\!]_\varsigma = \{\varsigma(t_i)\} = [\![t_i]\!]_s^\Gamma$ pour tout $i \in [1, n]$. Donc $[\![t]\!]_\varsigma = \{\varsigma(t)\} = [\![t]\!]_s^\Gamma$.

L'équivalence est donc vérifiée.

On considère maintenant $k > 0$ tel pour tout terme $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ayant strictement moins de k symboles définis l'équivalence est vérifiée.

- soit $v \in [\![t]\!]_\varsigma$, *i.e.* il existe une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ où $\varphi \in \mathcal{D}^n$ et $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ pour tout $i \in [1, n]$, et une substitution valeur ς' avec $\varsigma'(x) = \varsigma(x) \in [\![\tau_{@x}]\!]$ pour tout $x \in \mathcal{V}ar(t)$, telles que $v \in [\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_{\varsigma'}$ où z_s^{-p} est une variable fraîche avec $p := \triangleright^{\mathcal{P}}(\varsigma(t_1), \ldots, \varsigma(t_n))$ et $\varsigma'(z_s^{-p})$ est exempt de p. En notant, $u = t\left[t_{|\omega} \mapsto z\right]$, on a donc $v \in [\![u]\!]_{\varsigma'}$, et par induction, il existe un contexte d'évaluation valeur Γ tel que $v \in [\![r]\!]_s^\Gamma$ avec $z \mapsto w \in \Gamma$, où w est exempt de p. En notant $\Gamma' := \{t_{|\omega} \mapsto w\} \cup \Gamma$, on a donc $v \in [\![t]\!]_s^{\Gamma'} = [\![r]\!]_s^\Gamma$ et $[\![(\!|t_1, \ldots, t_n|\!)]\!]_s^{\Gamma'} = \{(\!|\varsigma(t_1), \ldots, \varsigma(t_n)|\!)\}$. Par conséquent, l'implication directe est bien vérifiée.
- Le raisonnement inverse donne l'implication indirecte.

Si $v \in [\![\tau]\!]_s$, par définition, il existe une subtitution valeur ς avec $\varsigma(x) \in [\![\tau_{@x}]\!]$, telle que $v \in [\![t]\!]_\varsigma$. On a donc bien un contexte d'évaluation Γ tel que τ est évaluable par Γ et $v \in [\![t]\!]_s^\Gamma$.

On suppose maintenant qu'il existe un contexte d'évaluation Γ tel que τ est évaluable par Γ. Soit $v[\![t]\!]_s^\Gamma$, par définition il existe un contexte d'évaluation valeur Γ' avec pour toute association $u \mapsto w \in \Gamma'$, $w \in [\![p]\!]$ où $u \mapsto p \in \Gamma$, tel que $v \in [\![t]\!]_s^{\Gamma'}$. De plus, comme τ est évaluable par Γ, pour toute association $\varphi_s^{!\mathcal{P}}(u_1, \ldots, u_n) \mapsto p \in \Gamma$, on a $[\![p]\!] \subseteq [\![x_s^{-q}]\!]$ où $q = \triangleright^{\mathcal{P}}(w_1, \ldots, w_n)$ avec $\{(\!|w_1, \ldots, w_n|\!)\} = [\![(\!|u_1, \ldots, u_n|\!)]\!]_s^{\Gamma'} \subseteq [\![(\!|u_1, \ldots, u_n|\!)]\!]_s^\Gamma$. Donc il existe une substitution valeur ς telle que $v \in [\![t]\!]_\varsigma$, avec $x \mapsto \varsigma(x) \in \Gamma'$ pour tout $x \in \mathcal{V}ar(t)$, d'où $\varsigma(x) \in [\![\tau_{@x}]\!]$. Par conséquent, on a bien $v \in [\![t]\!]_\varsigma \subseteq [\![\tau]\!]_s$. □

Afin de construire un contexte d'évaluation permettant de prouver ou de réfuter l'existence d'une valeur ne vérifiant pas certaines propriétés d'Exemption de Motif dans la sémantique du terme obtenu par inférence, on utilise la procédure de décision présentée dans la Figure 5.1.

La procédure de décision cherche donc à construire un contexte d'évaluation d'un terme de l'algèbre en imposant des contraintes à ses variables. En partant d'une version aliassée du membre droit de la règle considérée, on peut ainsi générer un contexte exprimant les contraintes induites par le non-respect de la post-condition du profil.

Axiomes et conjonction

$$\frac{}{x \mapsto u \setminus x_s^{-q} \vDash [\![x \mathbin{@} u]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset}\ (\mathbf{A}_\setminus) \qquad \frac{}{x \mapsto (u \times q) \downarrow_{\mathfrak{R}} \setminus \bot \vDash [\![x \mathbin{@} u]\!]_s \cap [\![q]\!] \neq \emptyset}\ (\mathbf{A}_\times)$$

$$\frac{\Gamma_1 \vDash P_1 \qquad \Gamma_2 \vDash P_2}{\Gamma_1 \sqcap \Gamma_2 \vDash P_1 \wedge P_2}\ (\wedge)$$

Décision constructeur

$$\frac{\Gamma \vDash [\![t_i]\!]_s \setminus [\![x_{s_i}^{-q}]\!] \neq \emptyset}{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset}\ (\mathbf{C}^{\mathbf{i}}_\setminus) \qquad \frac{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \cap [\![q]\!] \neq \emptyset}{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset}\ (\mathbf{C}^{\not<}_\setminus)$$

$$\frac{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \cap [\![q_i]\!] \neq \emptyset}{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \cap [\![q_1 + q_2]\!] \neq \emptyset}\ (\mathbf{C}^{+}_\times)$$

$$\frac{\Gamma \vDash \wedge_{i=1}^n [\![t_i]\!]_s \cap [\![q_i]\!] \neq \emptyset}{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \cap [\![\alpha(q_1, \ldots, q_n)]\!] \neq \emptyset}\ (\mathbf{C}^{=}_\times) \qquad \frac{\Gamma \vDash \wedge_{i=1}^n [\![t_i]\!]_s \cap [\![x_{s_i}^{-\bot}]\!] \neq \emptyset}{\Gamma \vDash [\![\alpha(t_1, \ldots, t_n)]\!]_s \cap [\![x_s^{-\bot}]\!] \neq \emptyset}\ (\mathbf{C}^{\mathbf{x}}_\times)$$

avec $\alpha \in \mathcal{C}$

Décision symbole défini

$$\frac{\Gamma; z \mapsto r \setminus S \vDash \wedge_{\Pi \subseteq \mathcal{P}}((\vee_{i=1}^n [\![t_i]\!]_s \setminus [\![x_{s_i}^{-\Pi_i^l}]\!] \neq \emptyset) \vee ([\![z \mathbin{@} y_s^{-\Pi^r}]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset))}{\Gamma; \varphi(t_1, \ldots, t_n) \mapsto r \setminus S \vDash [\![\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset}\ (\mathbf{F}_\setminus)$$

$$\frac{\Gamma; z \mapsto r \setminus S \vDash \wedge_{\Pi \subseteq \mathcal{P}}((\vee_{i=1}^n [\![t_i]\!]_s \setminus [\![x_{s_i}^{-\Pi_i^l}]\!] \neq \emptyset) \vee ([\![z \mathbin{@} y_s^{-\Pi^r}]\!]_s \cap [\![q]\!] \neq \emptyset))}{\Gamma; \varphi(t_1, \ldots, t_n) \mapsto r \setminus S \vDash [\![\varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)]\!]_s \cap [\![q]\!] \neq \emptyset}\ (\mathbf{F}_\times)$$

avec $\Pi^r = \sum_{p_1 * \cdots * p_n \mapsto q \in \Pi} q$ et $\Pi_j^l = \sum_{p_1 * \cdots * p_n \mapsto q \in \Pi} p_j$

FIGURE 5.1 – Procédure de décision pour construire un contexte d'évaluation suivant des contraintes d'Exemption de Motif sur les termes non-linéaires

Les axiomes $(\mathbf{A}_\setminus)$ et $(\mathbf{A}_\times)$ expriment ainsi que, si les contraintes d'instanciation de la variable sont satisfaisables, *i.e.* si la sémantique du motif associé à la variable par le contexte est non-vide, on peut toujours vérifier l'existence d'une valeur satisfaisant la propriété considérée. Ici, les prédicats que l'on cherche à vérifier sont généralement de la forme $[\![\rho]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$ avec ρ une version aliassée du membre droit $rs \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ de la règle, et q un motif additif linéaire et régulier. La vérification d'un tel prédicat peut se traduire en un prédicat de la forme $[\![\rho]\!]_s \cap [\![q]\!] \neq \emptyset$, dans

le cas où le motif q considéré filtre, en tête, l'instance correspondante de rs. On a donc un axiome pour chacune des formes de prédicats considérées, qui génèrent des contraintes d'instanciation de la variables sous la forme d'associations $x \mapsto r \setminus S$ avec r une somme de motifs quasi-symboliques linéaires et S une somme de variables annotées, *i.e.* $S = \sum_{q \in Q} x_s^{-q}$ pour un certain ensemble Q de motifs additifs linéaires et réguliers. Ces contraintes décrivent ainsi un contexte d'évaluation qui est propagé par le reste des règles de dérivations.

La règle de fusion des contextes d'évaluation ($\wedge$) exprime qu'étant donnés un contexte d'évaluation Γ_1 permettant de vérifier un prédicat P_1 et un contexte d'évaluation Γ_2 permettant de vérifier un prédicat P_2, la fusion $\Gamma_1 \sqcap \Gamma_2$ des contextes, si toutes les contraintes d'instanciation qu'elle exprime sont satisfaisables, permet de vérifier la conjonction des prédicats. Pour cela, on définit la fusion de Γ_1 et Γ_2 par la conjonction des contraintes d'instanciation :

$$\begin{aligned}\Gamma_1 \sqcap \Gamma_2 \quad = \quad & \{u \mapsto r \setminus S \mid u \mapsto r \setminus S \in \Gamma_1 . \nexists\, u \mapsto r' \setminus S' \in \Gamma_2\} \\ & \cup \{u \mapsto r \setminus S \mid u \mapsto r \setminus S \in \Gamma_2 . \nexists\, u \mapsto r' \setminus S' \in \Gamma_1\} \\ & \cup \{u \mapsto (r_1 \times r_2) \downarrow_{\mathfrak{R}} \setminus (S_1 + S_2) \mid u \mapsto r_1 \setminus S_1 \in \Gamma_1 \wedge u \mapsto r_2 \setminus S_2 \in \Gamma_2\}\end{aligned}$$

La procédure de décision fonctionne ensuite de manière inductive sur la forme de la version aliassée ρ considérée.

Pour un terme ayant un symbole constructeur comme symbole de tête, il existe une valeur de sa sémantique non-exempte d'un certain motif q, si et seulement si la non-Exemption de Motif peut se vérifier à sa racine ou dans un de ses sous-termes. La règle ($\mathbf{C}^{\mathbf{i}}_{\setminus}$) exprime ainsi l'application de la contrainte sur le i^{me} sous-terme, tandis que ($\mathbf{C}^{\prec}_{\setminus}$) exprime l'application de la contrainte à la racine. En appliquant la contrainte à la racine, on cherche une valeur de la sémantique filtrée par q, ce qui s'exprime par un prédicat de la forme $[\![\rho]\!]_s \cap [\![q]\!] \neq \emptyset$. On vérifie alors ces prédicats par induction sur q avec les règles ($\mathbf{C}^{+}_{\times}$), ($\mathbf{C}^{=}_{\times}$), et ($\mathbf{C}^{\mathbf{x}}_{\times}$), où la première exprime la décomposition sur l'opérateur de disjonction de motifs $+$, tandis que les deux autres expriment les règles de filtrage sur un motif constructeur et une variable, respectivement.

Pour un terme ayant un symbole défini comme symbole de tête, la règle ($\mathbf{F}_{\setminus}$) exprime la contrainte d'évaluation en fonction des profils de l'annotation du symbole défini : pour chaque combinaison de profils, soit la combinaison de profils n'est pas satisfaite par l'évaluation considérée, soit la propriété résultante des post-conditions des profils ne permet pas de satisfaire le prédicat. De façon similaire ($\mathbf{F}_{\times}$) exprime la même relation pour la forme conjonctive de prédicat. On peut également observer que ces règles résultent aussi en une contrainte d'évaluation du terme considéré, inférée via l'utilisation d'une variable fraîche z. En effet, il est nécessaire de vérifier que les contraintes considérées localement sont compatibles avec d'autres contraintes imposées à d'autres potentielles occurrences d'un même sous-terme dans le membre droit de la règle.

Enfin, lorsque toutes les contraintes d'instanciation données par le contexte sont satisfaisables, la procédure de décision permet de vérifier l'existence d'une valeur non exempte d'un certain profil dans la sémantique de la version aliassée considérée :

Proposition 5.23. *Étant donnés une sorte $s \in \mathcal{S}$, une version aliassée τ d'un terme $t \in \mathcal{T}_s(\mathcal{F}, \mathcal{X})$ et q un motif additif régulier et linéaire, on a $[\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$ si et seulement il existe Γ un contexte d'évaluation de t tel que :*

- *il existe une dérivation vérifiant $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$;*
- *pour toute association $u \mapsto r \setminus S \in \Gamma$, $[\![r]\!] \setminus [\![S]\!] \neq \emptyset$.*

Démonstration. Soient une sorte $s \in \mathcal{S}$, une version aliassée τ d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})s$ et q un motif additif régulier et linéaire.

Dans un premier temps, on considère Γ un contexte d'évaluation de t et P une conjonction de prédicats de la forme $[\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$ ou $[\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$, tels qu'il existe une dérivation vérifiant $\Gamma \vDash P$. On prouve par induction sur la dérivation que τ est évaluable par Γ et qu'on a $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, respectivement $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \cap [\![q]\!]$, pour chaque prédicat de P.

- pour les axiomes $(\mathbf{A}_\setminus)$ et $(\mathbf{A}_\times)$, avec $\Gamma = \{x \mapsto u \setminus x_s^{-q}\}$, on a $[\![x]\!]_s^\Gamma = [\![u \setminus x_s^{-q}]\!] = [\![x @ u]\!]_s \setminus [\![x_s^{-q}]\!]$, et respectivement, par préservation de la sémantique de $\mathfrak{R}$ (Proposition 5.4), avec $\Gamma = \{x \mapsto (u \times q) \downarrow_\mathfrak{R} \setminus \bot\}$, on a $[\![x]\!]_s^\Gamma = [\![(u \times q) \downarrow_\mathfrak{R} \setminus \bot]\!] = [\![u \times q]\!] = [\![x @ u]\!]_s \cap [\![q]\!]$.
- pour la règle de fusion $(\wedge)$, l'induction et la préservation de la sémantique de $\mathfrak{R}$ (Proposition 5.4) donne directement que pour chaque prédicat $[\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, respectivement $[\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$, de $P_1 \wedge P_2$, τ est évaluable par Γ et $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, respectivement $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \cap [\![q]\!]$.
- pour $(\mathbf{C}_\setminus^{\not\prec})$, on observe que pour toute valeur $v \in [\![\tau]\!]_s$ telle que $q \not\prec v$, v n'est pas exempte de q, *i.e.* $[\![\tau]\!]_s \cap [\![q]\!] \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$. Donc par induction τ est évaluable par Γ et $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$.
- pour $(\mathbf{C}_\setminus^{\mathbf{i}})$, par induction et avec Proposition 5.22, on a $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s$, et $[\![t_i]\!]_s^\Gamma \subseteq [\![\tau_i]\!]_s \setminus [\![x_{s_i}^{-q}]\!]$. Donc pour tout $v \in [\![c(t_1, \ldots, t_n)]\!]_s^\Gamma$, on a $v = c(v_1, \ldots, v_n)$ avec $v_i \notin [\![x_{s_i}^{-q}]\!]$, *i.e.* v_i non-exempt de q. Par conséquent v est également non-exempt de q, et on a donc bien $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$.
- pour $(\mathbf{C}_\times^{\mathbf{x}})$, par induction et avec Proposition 5.22, τ est évaluable par Γ et on a $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s = [\![\tau]\!]_s \cap [\![x_s^{-\bot}]\!]$.
- pour $(\mathbf{C}_\times^{=})$, par induction, on a pour tout $i \in [1, n]$, τ_i est évaluable par Γ et $[\![t_i]\!]_s^\Gamma \subseteq [\![\tau_i]\!]_s \cap [\![q_i]\!]$. Donc τ est évaluable avec Γ et, avec les Propositions 3.8 et 5.22, on a bien $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \cap [\![q]\!]$.
- pour $(\mathbf{C}_\times^{+})$, on rappelle, qu'avec la Proposition 3.8, on a $[\![q_1 + q_2]\!] = [\![q_1]\!] \cup [\![q_2]\!]$, d'où $[\![\tau]\!]_s \cap [\![q_i]\!] \subseteq [\![\tau]\!]_s \cap [\![q_1 + q_2]\!]$. Donc, par induction, τ est évaluable par Γ et on a bien $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \cap [\![q]\!]$.
- pour $(\mathbf{F}_\setminus)$, par induction, on a τ_i est évaluable par Γ pour tout $i \in [1, n]$, et, pour tout $\Pi \subseteq \mathcal{P}$, soit il existe j tel que $[\![t_j]\!]_s^\Gamma \subseteq [\![\tau_j]\!]_s \setminus [\![x_{s_j}^{-\Pi_j^l}]\!]$, soit $[\![r \setminus S]\!] \subseteq [\![x_s^{-\Pi^r}]\!] \setminus [\![x_s^{-q}]\!]$. Or, pour tout profil $\Pi \subseteq \mathcal{P}$ tel qu'il existe $(\!|v_1, \ldots, v_n|\!) \in [\![(\!|t_1, \ldots, t_n|\!)]\!]_s^\Gamma$ avec $\Pi = \{p_1 * \cdots * p_n \mapsto p \in \mathcal{P} \mid \forall i \in [1, n], v_i \text{ est exempt de } p_i\}$, on a, pour tout $i \in [1, n]$, $v_i \in [\![t_i]\!]_s^\Gamma$ et $v_i \in [\![x_{s_i}^{-\Pi_i^l}]\!]$. Donc, pour de tels profils Π, dont l'ensemble vide, $[\![t_i]\!]_s^\Gamma \not\subseteq [\![t_i]\!]_s \setminus [\![x_{s_i}^{-\Pi_i^l}]\!]$ d'où, $[\![r \setminus S]\!] \subseteq [\![x_s^{-\Pi^r}]\!] \setminus [\![x_s^{-q}]\!]$. On a donc $[\![r \setminus S]\!] \cap [\![x_s^{-q}]\!] = \emptyset$ et d'après la Proposition 5.22, $[\![t]\!]_s^\Gamma = [\![r \setminus S]\!] \subseteq [\![\tau]\!]_s$. On a donc bien τ évaluable par Γ et $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$.
- De même, l'induction est également vérifiée pour $(\mathbf{F}_\times)$.

La propriété est donc vérifiée par induction, et s'il existe Γ un contexte d'évaluation de t et une dérivation vérifiant $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, alors on a $[\![t]\!]_s^\Gamma \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$. Si, pour toute association $u \mapsto r \setminus S \in \Gamma$, $[\![r]\!] \setminus [\![S]\!] \neq \emptyset$, on a donc $[\![t]\!]_s^\Gamma \neq \emptyset$, d'où $[\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$.

On montre maintenant par induction sur la forme de t que pour tout contexte d'évaluation valeur Γ' de t tel que τ est évaluable par Γ' et $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, respectivement $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \cap [\![q]\!]$, il existe Γ un contexte d'évaluation de t tel qu'il existe une dérivation vérifiant $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, respectivement $\Gamma \vDash [\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$, et que pour toute association $u \mapsto r \setminus S \in \Gamma$, il existe $v \in [\![r \setminus S]\!]$ tel que $u \mapsto v \in \Gamma'$.

- pour $t = x \in \mathcal{X}$, on a $\tau = x @ u$. Avec les axiomes $(\mathbf{A}_\setminus)$ et $(\mathbf{A}_\times)$, on a donc $x \mapsto u \setminus x_s^{-q} \vDash [\![x @ u]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, respectivement $x \mapsto (u \times q) \downarrow_\mathfrak{R} \setminus \bot \vDash [\![x @ u]\!]_s \cap [\![q]\!] \neq \emptyset$. De plus par

définition, avec $\Gamma = \{x \mapsto u \setminus x_s^{-q}\}$, on a $[\![x]\!]_s^\Gamma = [\![u \setminus x_s^{-q}]\!] = [\![x \mathbin{@} u]\!]_s \setminus [\![x_s^{-q}]\!]$, et, par préservation de la sémantique de $\mathfrak{R}$ (Proposition 5.4), avec $\Gamma = \{x \mapsto (u \times q) \downarrow_{\mathfrak{R}} \setminus \bot\}$, on a $[\![x]\!]_s^\Gamma = [\![(u \times q) \downarrow_{\mathfrak{R}} \setminus \bot]\!] = [\![u \times q]\!] = [\![x \mathbin{@} u]\!]_s \cap [\![q]\!]$.

- pour $t = c(t_1, \ldots, t_n)$, on a $\tau = c(\tau_1, \ldots, \tau_n)$ avec, pour tout $i \in [1, n]$, τ_i une version aliassée de t_i.
  - ▶ On a pour tout contexte d'évaluation valeur Γ' de t tel que $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \cap [\![q]\!]$, $[\![t]\!]_s^{\Gamma'} = \{c(v_1, \ldots, v_n)\}$ avec $\{v_i\} = [\![t_i]\!]_s^{\Gamma'} \subseteq [\![\tau_i]\!]_s$ pour tout $i \in [1, n]$, et $q \not\preccurlyeq c(v_1, \ldots, v_n)$. On procède alors par induction sur la forme de q :
    - — pour $q = x_s^{-\bot}$, on a alors, pour tout $i \in [1, n]$, $v_i \in [\![\tau_i]\!]_s = [\![\tau_i]\!]_s \cap [\![x_{s_i}^{-\bot}]\!]$. Par induction, pour tout $i \in [1, n]$, il existe donc Γ_i tel qu'il existe une dérivation vérifiant $\Gamma_i \vDash [\![\tau_i]\!]_s \cap [\![x_{s_i}^{-\bot}]\!] \neq \emptyset$ et que pour toute association $u \mapsto r \setminus S \in \Gamma_i$, il existe $v \in [\![r \setminus S]\!]$ tel que $u \mapsto v \in \Gamma'$. Donc, en notant $\Gamma = \Gamma_1 \sqcap \cdots \sqcap \Gamma_n$, via la règle $(\wedge)$, on a une dérivation vérifiant $\Gamma \vDash [\![\tau_i]\!]_s \cap [\![x_s^{-\bot}]\!] \neq \emptyset$, et, par préservation de la réécriture via $\mathfrak{R}$, pour toute association $u \mapsto r \setminus S \in \Gamma$, il existe $v \in [\![r \setminus S]\!]$ tel que $u \mapsto v \in \Gamma'$.
    - — pour $q = c(q_1, \ldots, q_n)$, on a alors, pour tout $i \in [1, n]$, $v_i \in [\![\tau_i]\!]_s = [\![\tau_i]\!]_s \cap [\![q_i]\!]$. Avec le même raisonnement, on peut construire un contexte d'évaluation Γ vérifiant la propriété voulue.
    - — pour $q = q_1 + q_2$, on a alors $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \cap [\![q_1]\!]$ ou $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \cap [\![q_2]\!]$. Par induction sur la forme de q, il existe donc un contexte d'évaluation Γ vérifiant la propriété voulue.
  - ▶ On a pour tout contexte d'évaluation valeur Γ' de t tel que $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, $[\![t]\!]_s^{\Gamma'} = \{c(v_1, \ldots, v_n)\}$ avec $\{v_i\} = [\![t_i]\!]_s^{\Gamma'} \subseteq [\![\tau_i]\!]_s$ pour tout $i \in [1, n]$, et comme $v := c(v_1, \ldots, v_n)$ n'est pas exempt de q, il existe une position $\omega \in \mathcal{P}os(t)$ telle que $q \not\preccurlyeq v$. Si $\omega = \epsilon$, on a $q \not\preccurlyeq v$, et comme dans le cas précédent, on peut construire un contexte d'évaluation Γ tel qu'il existe une dérivation vérifiant $\Gamma \vDash [\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$, donc avec la règle $(\mathbf{C}_{\setminus}^{\not\preccurlyeq})$, on a $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$. Sinon $\omega = j.\omega'$ avec $j \in [1, n]$, et on a alors $[\![t_j]\!]_s^{\Gamma'} \subseteq= [\![\tau_j]\!]_s \setminus [\![x_{s_j}^{-q}]\!]$. Par induction, il existe alors Γ tel qu'il existe une dérivation vérifiant $\Gamma \vDash [\![\tau_j]\!]_s \setminus [\![x_{s_j}^{-q}]\!] \neq \emptyset$, donc, avec la règle $(\mathbf{C}_{\setminus}^{\mathbf{i}})$, on a $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$.
- pour $t = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$, on a $\tau = \varphi_s^{!\mathcal{P}}(\tau_1, \ldots, \tau_n)$, avec, pour tout $i \in [1, n]$, τ_i une version aliassée de t_i.
  - ▶ On considère un contexte d'évaluation Γ' de t tel que $[\![t]\!]_s^{\Gamma'} = \{v\} \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$. On note $\{(\!|v_1, \ldots, v_n|\!)\} = [\![(\!|\tau_1, \ldots, \tau_n|\!)]\!]_s^{\Gamma'}$ et comme τ est évaluable par Γ', on a v exempt de $p := \triangleright^{\mathcal{P}}(v_1, \ldots, v_n)$ et pour tout $\Pi \subseteq \mathcal{P}$ tel que pour tout $i \in [1, n]$ $v_i \in [\![x_{s_i}^{-\Pi_i^l}]\!]$, on a $[\![x_s^{-\Pi^r}]\!] \subseteq [\![x_s^{-p}]\!]$. On note $r := \prod_\Pi [\![x_s^{-\Pi^r}]\!]$ avec $\Pi \subseteq \mathcal{P}$ tel que pour tout $i \in [1, n]$ $v_i \in [\![x_{s_i}^{-\Pi_i^l}]\!]$, et avec les règles $(\mathbf{A}_{\setminus})$ et $(\wedge)$, on a donc une dérivation vérifiant $z \mapsto r \setminus x_s^{-q} \vDash \vee_\Pi [\![z \mathbin{@} y_s^{-\Pi^r}]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, et $v \in [\![r \setminus x_s^{-q}]\!]$. Pour tout $\Pi \subseteq \mathcal{P}$ tel qu'il existe $j \in [1, n]$ avec $v_j \notin [\![x_{s_j}^{-\Pi_j^l}]\!]$, par induction, il existe alors Γ_Π tel qu'il existe une dérivation vérifiant $\Gamma_\Pi \vDash [\![\tau_j]\!]_s \setminus [\![x_{s_j}^{-\Pi_j}]\!] \neq \emptyset$, et que pour toute association $u \mapsto r \setminus S \in \Gamma_\Pi$, il existe $v \in [\![r \setminus S]\!]$ tel que $u \mapsto v \in \Gamma'$. On note donc $\Gamma := \bigsqcap_\Pi \Gamma_\Pi$ via la règle $(\wedge)$, on a une dérivation vérifiant $\Gamma \vDash \wedge_\Pi (\vee_{i=1}^n [\![\tau_i]\!]_s \setminus [\![x_{s_i}^{-\Pi_i^l}]\!] \neq \emptyset$, avec $\Pi \subseteq \mathcal{P}$ tel qu'il existe $j \in [1, n]$ avec $v_j \notin [\![x_{s_j}^{-\Pi_j^l}]\!]$, et, par préservation de la réécriture via $\mathfrak{R}$, pour toute association $u \mapsto r' \setminus S \in \Gamma$, il existe $v \in [\![r' \setminus S]\!]$ tel que $u \mapsto v \in \Gamma'$. Enfin,

on fusionne les dérivations avec $(\wedge)$, et on applique $(\mathbf{F}_\setminus)$ pour obtenir une dérivation vérifiant $\Gamma; \tau \mapsto r \setminus x_s^{-q} \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$.

- Le même raisonement permet de vérifier l'induction pour le prédicat $[\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$.

La propriété est donc vérifiée par induction, et s'il existe $v \in [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, respectivement $v \in [\![\tau]\!]_s \cap [\![q]\!]$, d'après la Proposition 5.22, alors il existe un contexte (valeur) Γ' tel que τ est évaluable par Γ' et $[\![t]\!]_s^{\Gamma'} \subseteq [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!]$, respectivement $[\![t]\!]_s^{\Gamma'} [\![\tau]\!]_s \cap [\![q]\!]$. Il existe donc Γ un contexte d'évaluation de t tel qu'il existe une dérivation vérifiant $\Gamma \vDash [\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, respectivement $\Gamma \vDash [\![\tau]\!]_s \cap [\![q]\!] \neq \emptyset$. □

Étant donnée τ une version aliassée d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, cette procédure de décision permet donc, pour tout motif additif linéaire et régulier q, de construire un contexte d'évaluation évaluant τ tel que toute valeur de l'évaluation de t suivant Γ n'est pas exempte de q. Si cette évaluation est non-vide, *i.e.* si la sémantique de toutes les association de Γ est non-vide, on peut ainsi vérifier l'existence d'une valeur non-exempte de q dans la sémantique $[\![\tau]\!]_s$. Inversement, comme il n'y a qu'un nombre fini de dérivations possibles vérifiant $[\![\tau]\!]_s \setminus [\![x_s^{-q}]\!] \neq \emptyset$, si aucune dérivation ne fournit un contexte d'évaluation de t dont la sémantique de toutes les associations est non-vide, on peut alors conclure que τ est exempt de q modulo $[\![]\!]_s$.

Exemple 5.12. *On considère l'algèbre de termes définie par la signature Σ_{nl} présentée dans l'Exemple 3.7, ainsi que les symboles définis et le système $\mathcal{R}$ introduit dans l'Exemple 5.4. On rappelle notamment que le symbole $h^{!\mathcal{P}_h}$ est annoté avec $\mathcal{P}_h = \{b * \bot \mapsto c(a,b), \bot * a \mapsto c(a,b)\}$.*

On peut maintenant utiliser la procédure de décision pour vérifier que, comme montré dans l'Exemple 5.7, $h(x,x)$ est bien exempt de $c(a,b)$ modulo $[\![]\!]_s$. On considère la version aliassée de $h(x,x)$, et on observe qu'il n'existe qu'une seule dérivation vérifiant $[\![h(x \mathbin{@} y_{\mathsf{S2}}^{-\bot}, x \mathbin{@} y_{\mathsf{S2}}^{-\bot})]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma_x \vDash [\![x \mathbin{@} y_{\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![x_{\mathsf{S2}}^{-a}]\!] \neq \emptyset}(\mathbf{A}_\setminus) \quad \dfrac{}{\Gamma'_x \vDash [\![x \mathbin{@} y_{\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![x_{\mathsf{S2}}^{-b}]\!] \neq \emptyset}(\mathbf{A}_\setminus)}{\Gamma_x \sqcap \Gamma'_x; z \mapsto x_{\mathsf{S1}}^{-\bot} \setminus x_{\mathsf{S1}}^{-c(a,b)} \vDash [\![x \mathbin{@} y_{\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![x_{\mathsf{S2}}^{-a}]\!] \neq \emptyset \wedge [\![x \mathbin{@} y_{\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![x_{\mathsf{S2}}^{-b}]\!] \neq \emptyset \wedge [\![z \mathbin{@} x_{\mathsf{S1}}^{-\bot}]\!]_s \setminus [\!]\!] \neq \emptyset}(\wedge) \quad \dfrac{}{z \mapsto x_{\mathsf{S2}}^{-\bot} \setminus x_{\mathsf{S2}}^{-b} \vDash [\![z \mathbin{@} x_{\mathsf{S1}}^{-\bot}]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}(\mathbf{A}_\setminus)}{\Gamma_x \sqcap \Gamma'_x; h(x,x) \mapsto x_{\mathsf{S1}}^{-\bot} \setminus x_{\mathsf{S1}}^{-c(a,b)} \vDash [\![h^{!\mathcal{P}_h}(x \mathbin{@} y_{\mathsf{S2}}^{-\bot}, x \mathbin{@} y_{\mathsf{S2}}^{-\bot})]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}(\mathbf{F}_\setminus)$$

Avec $\Gamma_x = \{x \mapsto y_{\mathsf{S2}}^{-\bot} \setminus x_{\mathsf{S2}}^{-a}\}$ et $\Gamma'_x = \{x \mapsto y_{\mathsf{S2}}^{-\bot} \setminus x_{\mathsf{S2}}^{-b}\}$, d'où $\Gamma_x \sqcap \Gamma'_x = \{x \mapsto y_{\mathsf{S2}}^{-\bot} \setminus (x_{\mathsf{S2}}^{-a} + x_{\mathsf{S2}}^{-b})\}$. Comme $[\![y_{\mathsf{S2}}^{-\bot}]\!] = \{a, b\}$, on a clairement $[\![y_{\mathsf{S2}}^{-\bot}]\!] \setminus [\![x_{\mathsf{S2}}^{-a} + x_{\mathsf{S2}}^{-b}]\!] = \emptyset$. Donc on peut conclure que $[\![h(x,x)]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] = \emptyset$, i.e. $h(x,x)$ est exempt de $c(a,b)$ modulo $[\![]\!]_s$.

Dans l'exemple précédent, on a facilement pu observer que la contrainte d'instanciation exprimée par l'association du contexte d'évaluation n'était pas satisfaisable. Dans le cas général, la procédure de décision génère des contraintes d'évaluation sous la forme d'associations $u \mapsto r \setminus S$ avec r une somme de motifs quasi-symboliques linéaires et S une somme de variables annotées. On propose dans la sous-section suivante de vérifier la satisfaisabilité d'une telle contrainte en vérifiant que la différence de sémantiques $[\![r]\!] \setminus [\![S]\!]$ est non-vide.

5.3.3 Satisfaction de contraintes d'évaluation

Afin d'établir la satisfaction d'un profil, la procédure de décision présentée permet donc d'exprimer la post-condition du profil sous la forme de contraintes d'instanciation du membre droit de la règle considérée. Ces contraintes sont décrites par un contexte d'évaluation contenant des associations $u \mapsto r \setminus \sum_{q \in Q} x_s^{-q}$, et sont donc satisfaisables si et seulement si la sémantique $[\![r \setminus \sum_{q \in Q} x_s^{-q}]\!]$ est non-vide.

Vérifier la satisfaisabilité de ces contraintes revient donc à démontrer l'existence d'une valeur $v \in [\![r]\!]$ non-exempte de q, pour tout $q \in Q$. On propose donc d'utiliser la notion de graphe d'atteignabilité, dont la construction est présentée en Section 4.1, pour construire une telle valeur. En effet, les nœuds composant le graphe d'atteignabilité de r permettent d'exprimer la forme de tous les sous-termes de sa sémantique, et donc de vérifier l'existence d'un sous-terme filtré par chaque motif $q \in Q$.

La représentation graphique de la forme des sous-termes fournie par le graphe d'atteignabilité permet ainsi de partitionner les contraintes considérées, entre celles qui s'appliquent directement et celles qui s'appliqueront récursivement :

Lemme 5.24. *Étant donné une sorte $s \in \mathcal{S}$, un motif quasi-symbolique linéaire p, un graphe $G = (V, E)$ d'atteignabilité de p et un ensemble fini Q de motifs quasi-additifs linéaires et régulier. On a $[\![p]\!] \setminus [\![\sum_{q\in Q} x_s^{-q}]\!] \neq \emptyset$ si et seulement si il existe des nœuds $u_1, \ldots, u_n \in V$ et une $n+1$-partition $\{q_1, \ldots, q_n\} \uplus Q_1 \uplus \cdots \uplus Q_n = Q$ tels que :*

- *il existe des chemins dans G de p à $u_1, \ldots, u_n$, tels que pour tout $i, j \in [1, n]$ avec $i \neq j$, le sous-chemin de p jusqu'au dernier nœud variable avant u_i, ou jusque u_i si u_i est un nœud variable ou que le chemin ne passe pas par un nœud variable, n'est pas un préfixe du chemin de p à u_j ;*
- *$[\![u_i \times q_i]\!] \setminus [\![\sum_{q\in Q_i} x_s^{-q}]\!] \neq \emptyset$, pour tout $i \in [1, n]$.*

Démonstration. Soient une sorte $s \in \mathcal{S}$, un motif quasi-symbolique linéaire p, un graphe $G = (V, E)$ d'atteignabilité de p et un ensemble fini Q de motifs quasi-additifs linéaires et régulier.

- On suppose qu'il existe des nœuds $u_1, \ldots, u_n \in V$ et une $n+1$-partition $\{q_1, \ldots, q_n\} \uplus Q_1 \uplus \cdots \uplus Q_n = Q$ vérifiant le Lemme. On peut construire un terme de $[\![p]\!] \setminus [\![\sum_{q\in Q} x_s^{-q}]\!]$ en remplaçant chaque nœud u_i par une valeur de $[\![u_i \times q_i]\!] \setminus [\![\sum_{q\in Q_i} x_s^{-q}]\!]$ et les nœuds (quasi-) variables restant par une valeur quelconque de leur sémantique. Comme les chemins dans G de p à $u_1, \ldots, u_n$ n'entrent pas en conflit, le graphe ainsi construit décrit une valeur de la sémantique de p non-exempte de q, pour tout $q \in Q$.
- On suppose maintenant qu'il existe une valeur $v \in [\![p]\!] \setminus [\![\sum_{q\in Q} x_s^{-q}]\!]$ et on note $\{q_1, \ldots, q_m\} := Q$. Pour tout $i \in [1, m]$, comme v n'est exempt de q_i, il existe une position $\omega_i \in \mathcal{P}os(t)$ telle que $q_i \not\ll v_{|\omega_i}$. On peut donc construire une $n+1$-partition de l'ensemble des positions $\{\omega_1, \ldots, \omega_m\} : \{\omega'_1, \ldots, \omega'_n\} \uplus \Omega_1 \uplus \cdots \uplus \Omega_n$ telle que pour tout $i \in [1, n]$, pour tout $\omega \in \Omega_i$ $\omega'_i < \omega$, et pour tout $j \neq i$ $\omega'_i \not< \omega'_j$. Par construction du graphe d'atteignabilité, les n positions $\omega'_1, \ldots, \omega'_n$ décrivent ainsi n chemins sans conflits vers des nœuds $u_1, \ldots, u_n$ du graphe d'atteignabilité de p. De plus, pour tout $i \in [1, n]$, $[\![u_i \times q'_i]\!] \setminus [\![\sum_{q\in Q_i} x_{s_i}^{-q}]\!] \neq \emptyset$ avec q'_i le motif correspondant à la position ω'_i et Q_i l'ensemble des motifs correspondant à l'ensemble des positions Ω_i.

□

Exemple 5.13. *On considère l'algèbre de termes définie par la signature Σ_{list} présentée dans l'Exemple 3.5. On étudie le terme $p = cons(e^{-\bot}_{\mathsf{Expr}}, l^{-p_{flat}}_{\mathsf{List}})$, avec $p_{flat} = cons(lst(l_1), l_2)$ et on montre que*

$$[\![p]\!] \setminus [\![x^{-p_1}_{\mathsf{List}} + x^{-p_2}_{\mathsf{List}} + x^{-p_3}_{\mathsf{List}}]\!] \neq \emptyset$$

avec $p_1 = lst(nil)$, $p_2 = int(i)$ et $p_3 = cons(e, nil)$.

À partir du graphe construit dans l'Example 4.5 et du graphe d'atteignabilité de $l^{-p_{flat}}_{\mathsf{List}}$ obtenu dans l'Exemple 4.7, on construit de façon similaire le graphe d'atteignabilité de p :

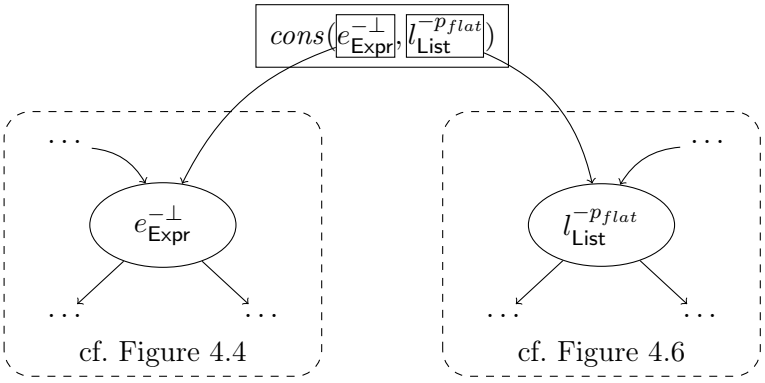
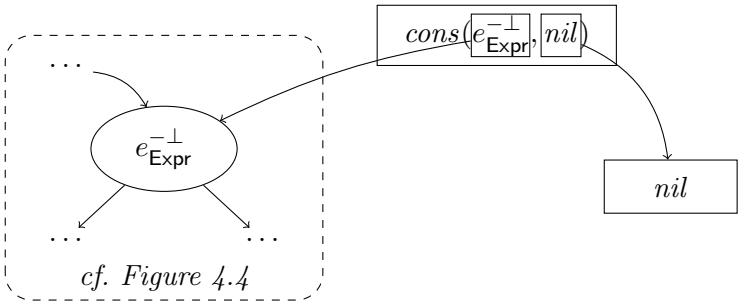


FIGURE 5.2 – Graphe d'atteignabilité de $cons(e^{-\bot}_{\mathsf{Expr}}, l^{-p_{flat}}_{\mathsf{List}})$ dans l'algèbre définie par Σ_{list}

Pour vérifier la relation recherchée en se basant sur le Lemme précédent, on calcule les conjonctions entre les nœuds du graphe d'atteignabilité et les motifs p_1, p_2, p_3. On peut ainsi commencer par rejeter la conjonction $p \times p_3$ qui se réduit à $cons(e^{-\bot}_{\mathsf{Expr}}, nil)$ (en simplifiant les alias inutiles) via le système $\mathfrak{R}$. Toutes les instances de la conjonction sont alors des listes contenant un seul élément, et ne peuvent donc pas contenir à la fois un sous-terme filtré par p_1 et un sous-terme filtré par p_2. On peut l'observer en construisant le graphe d'atteignabilité de $cons(e^{-\bot}_{\mathsf{Expr}}, nil)$:



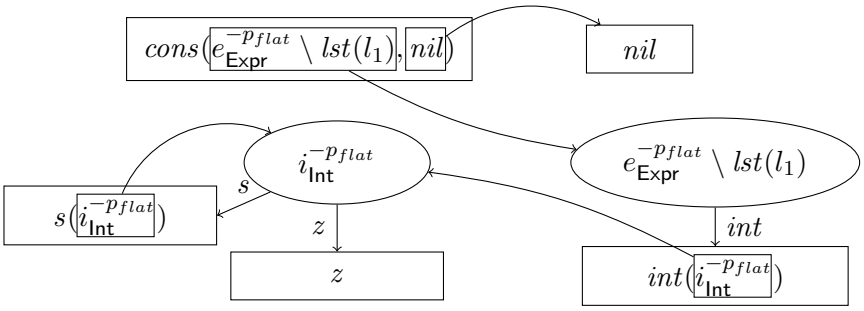
Les seuls nœuds de ce graphe, dont les conjonctions avec $int(i)$ et $lst(nil)$ ne sont pas vides, sont le nœud $e^{-\bot}_{\mathsf{Expr}}$ et ses descendants directs. Comme, il n'existe pas deux chemins depuis la racine jusque $e^{-\bot}_{\mathsf{Expr}}$ dont l'un n'est pas un préfixe de l'autre, et qu'un terme filtré par p_2 ou p_3 est exempt de l'autre, le Lemme précédent garantit qu'il n'existe pas d'instance de $cons(e^{-\bot}_{\mathsf{Expr}}, nil)$ non-exempte de p_2 et p_3.

On a ainsi formellement rejeté la conjonction $p \times p_3$, et on s'intéresse, maintenant, aux nœuds suivants. On peut considérer d'une part les conjonctions $e^{-\bot}_{\mathsf{Expr}} \times int(i)$ et $e^{-\bot}_{\mathsf{Expr}} \times lst(nil)$ et d'autre part $l^{-p_{flat}}_{\mathsf{List}} \times cons(e, nil)$:

- *si on considère la conjonction $e^{-\bot}_{\mathsf{Expr}} \times int(i)$, il faudra trouver une conjonction entre un nœud du graphe d'atteignabilité de $l^{-p_{flat}}_{\mathsf{List}}$ et $lst(nil)$, dont la sémantique est non-vide. Or ce graphe ne contenant pas le symbole lst, une telle conjonction aura forcément une sémantique vide.*
- *on considère donc les conjonctions $e^{-\bot}_{\mathsf{Expr}} \times lst(nil)$ et $l^{-p_{flat}}_{\mathsf{List}} \times cons(e, nil)$. Toutes les instances de $lst(nil)$ étant exemptes de $int(i)$, il reste alors à montrer que $[\![l^{-p_{flat}}_{\mathsf{List}} \times cons(e, nil)]\!] \setminus [\![x^{-p_2}_{\mathsf{List}}]\!] \neq \emptyset$.*

On recommence donc l'opération sur la conjonction $l^{-p_{flat}}_{\mathsf{List}} \times cons(e, nil)$, qui se réduit à

$cons(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1), nil)$ (en éliminant les alias inutiles) via le système $\mathfrak{R}$. On construit son graphe d'atteignabilité :



On peut voir grâce à ce graphe que $int(z) \in [\![(e_{\mathsf{Expr}}^{-p_{flat}} \setminus lst(l_1)) \times int(i)]\!]$, donc on peut enfin conclure avec le Lemme précédent qu'on a bien :

$$[\![p]\!] \setminus [\![x_{\mathsf{List}}^{-p_1} + x_{\mathsf{List}}^{-p_2} + x_{\mathsf{List}}^{-p_3}]\!] \neq \emptyset$$

De plus, en suivant ainsi la logique on peut construire un motif dont les instances permettent, en effet, de vérifier ce résultat :

- *les instances de $cons(int(i_{\mathsf{Int}}^{-p_{flat}}), nil)$ ne sont pas exemptes de p_2 ;*
- *les instances de $cons(int(i_{\mathsf{Int}}^{-p_{flat}}), nil)$ ne sont donc exemptes ni de p_2, ni de p_3 ;*
- *les instances de $cons(lst(nil), cons(int(i_{\mathsf{Int}}^{-p_{flat}}), nil)))$ ne sont donc exemptes ni de p_1, ni de p_2, ni de p_3.*

Étant donnés un motif quasi-symbolique linéaire r et un ensemble fini Q de motifs additifs linéaires et réguliers, en se basant sur le Lemme précédent, on propose ainsi de vérifier que $[\![p]\!] \setminus [\![\sum_{q \in Q} x_s^{-q}]\!] \neq \emptyset$ avec l'algorithme `checkInstance` présenté en Figure 5.3.

Étant donnés une règle $ls \twoheadrightarrow rs$ et un profil $p_1 * \cdots * p_n \mapsto q$, avec l'approche d'inférence montrée dans la Proposition 5.21, on peut prouver la satisfaction du profil en vérifiant que pour les formes aliassées ρ du membre droit rs, on a $[\![\rho]\!]_s \subseteq [\![x_s^{-q}]\!]$. La procédure de décision présentée en Figure 5.1 permet d'exprimer la non-satisfaction de cette post-condition sous la forme de contraintes d'instanciation des variables du membre droit de la règle. Enfin, l'algorithme `checkInstance` permet de vérifier la satisfaisabilité de ces contraintes et donc de conclure quant à la satisfaction des profils. Ces différents mécanismes définissent ainsi une méthode d'analyse statique vérifiant que le CBTRS considéré préserve la sémantique substitutive.

5.4 Application à l'analyse statique de systèmes non-linéaires

Les différents résultats présentés dans la Section précédente permettent donc de définir une méthode d'analyse statique des systèmes non-linéaires, visant à vérifier que les annotations choisies pour décorer les symboles définis sont en accord avec la relation de réécriture induite du CBTRS considéré. Pour cela, on se repose sur la notion de sémantique substitutive (Définition 5.7) et sa préservation par le CBTRS (Définition 5.8).

5.4.1 Récapitulatif de la méthode d'analyse

Pour prouver qu'un CBTRS donné préserve la sémantique substitutive, la Proposition 5.19 établit qu'il est nécessaire et suffisant de montrer que chaque règle satisfait par substitution tous

Fonction `checkInstance`(p, Q, R, S)
  Entrées : p : motif quasi-symbolique linéaire,
    Q : ensemble fini de motifs additifs linéaires,
    R : ensemble de motifs quasi-symboliques linéaires explorables,
    S : ensemble de motifs quasi-symboliques linéaires déjà explorés
  Résultat : $True$ si $[\![p]\!] \setminus [\![\sum_{q\in Q} x_s^{-q}]\!] \neq \emptyset$; $False$ sinon
  si $p \in S$ **alors**
    si $R \neq \emptyset$ **alors**
      retourner `checkInstance` $(u, Q, R \setminus \{u\}, \emptyset)$ avec $u \in R$
    retourner $False$
  pour $q \in Q$ **faire**
    $r \longleftarrow (p \times q)\downarrow_{\mathfrak{R}}$
    si $r \neq \bot$ **alors**
      pour $i = 1$ **à** n *avec* $r = \sum_{i=1}^{n} r_i$ **faire**
        si `checkInstance`$(r_i, Q \setminus \{q\}, R, \emptyset)$ **alors**
          retourner $True$
  filtre p **:**
    avec $c(p_1, \ldots, p_n) \Rightarrow$
      $R' \longleftarrow R \cup \{p_1, \ldots, p_n\}$
      pour $i = 1$ **à** n **faire**
        si `checkInstance`$(p_i, Q, R' \setminus \{p_i\}, S \cup \{p\})$ **alors**
          retourner $True$
    avec $x_s^{-r} \setminus t \Rightarrow$
      pour $c \in \mathcal{C}_s$ **faire**
        $(s_1, \ldots, s_n) \longleftarrow \mathcal{D}om(c)$
        $r \longleftarrow (p \times c(z_1{}_{s_1}^{-\bot}, \ldots, z_n{}_{s_n}^{-\bot}))\downarrow_{\mathfrak{R}}$
        si $r \neq \bot$ **alors**
          pour $i = 1$ **à** n *avec* $r = \sum_{i=1}^{n} r_i$ **faire**
            si `checkInstance`$(r_i, Q, R, S \cup \{p\})$ **alors**
              retourner $True$
  retourner $False$

FIGURE 5.3 – Algorithme `checkInstance`

les profils du symbole de tête de son membre gauche. On utilise alors la Proposition 5.21 pour vérifier que ces profils sont bien satisfaits.

On peut donc appliquer l'analyse présentée dans la Section précédente pour garantir que la sémantique substitutive est préservée par la relation de réécriture induite du CBTRS considéré :

Theorem 5.25. *Étant donné un CBTRS confluent $\mathcal{R}$, si pour toute règle $f^{!\mathcal{P}}(ls_1, \ldots, ls_n) \twoheadrightarrow rs \in \mathcal{R}$ et pour tout profil $p_1 * \cdots * p_n \mapsto p \in \mathcal{P}$, où $p_1, \ldots, p_n, p$ sont des motifs réguliers linéaires, en notant $v := (\!|ls_1 \times z_1{}_{s_1}^{-p_1}, \ldots, ls_n \times z_n{}_{s_n}^{-p_n}|\!)\downarrow_{\mathfrak{R}}$, on a :*

- $v = \bot$;
- *ou* $v = (\!|\lambda_1^1, \ldots, \lambda_n^1|\!) + (\!|\lambda_1^2, \ldots, \lambda_n^2|\!) + \cdots + (\!|\lambda_1^m, \ldots, \lambda_n^m|\!)$ *avec chaque* $\lambda_i^j, j \in [1, m]$ *une version aliassée de* $ls_i, i \in [1, n]$, *et*
  - *soit il existe* $x \in \mathcal{V}ar(ls)$ *telle que* $(\!|\lambda_1^k, \ldots, \lambda_n^k|\!)_{@x}\downarrow_{\mathfrak{R}} = \bot$;

- *soit* $[\![\sigma^{@(\!|\lambda_1^k,\ldots,\lambda_n^k|\!)}_{(\!|ls_1,\ldots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$.

alors on a pour tout terme clos $t, v \in \mathcal{T}(\mathcal{F})$ *:*

1. *Si* $v = t\downarrow_{\mathcal{R}}$*, alors :*
   1.1. $[\![v]\!]_s \subseteq [\![t]\!]_s$ *;*
   1.2. *pour tout motif* q *tel que* t *est exempt de* q *modulo* $[\![]\!]_s$*,* v *est exempt de* q *modulo* $[\![]\!]_s$ *;*
2. *Si* $\mathcal{R}$ *est complet et* $v = t\downarrow_{\mathcal{R}}$*, alors* v *est une valeur et, pour tout motif* q *tel que* t *est exempt de* q*,* $q \not\preccurlyeq v_{|\omega}$ *pour toute position* $\omega \in \mathcal{P}os(v)$*.*

Démonstration. Soit un CBTRS $\mathcal{R}$ vérifiant toutes les conditions, d'après les Propositions 5.19 et 5.21, $\mathcal{R}$ préserve la sémantique substitutive. Soient des termes clos $t, v \in \mathcal{T}(\mathcal{F})$ avec $v = t\downarrow_{\mathcal{R}}$, on prouve par induction sur la longueur de la $\mathcal{R}$-dérivation de t à v que $[\![v]\!]_s \subseteq [\![t]\!]_s$. Si la dérivation est vide, on a $t = v$ d'où $[\![v]\!]_s \subseteq [\![t]\!]_s$.

On suppose maintenant que pour tout u tel que $v = u\downarrow_{\mathcal{R}}$, avec une $\mathcal{R}$-dérivation de u à v strictement plus courte que la $\mathcal{R}$-dérivation de t à v, on a $[\![v]\!]_s \subseteq [\![u]\!]_s$. Comme $\mathcal{R}$ préserve la sémantique, d'après la Proposition 5.16, il existe u tel que $t \Longrightarrow_{\mathcal{R}} \cdots \Longrightarrow^*_{\mathcal{R}} u$ et $[\![u]\!]_s \subseteq [\![t]\!]_s$. De plus par confluence, on a $u\downarrow_{\mathcal{R}} = t\downarrow_{\mathcal{R}} = v$, donc par hypothèse d'induction, on a $[\![v]\!]_s \subseteq [\![u]\!]_s$, d'où $[\![v]\!]_s \subseteq [\![t]\!]_s$.

Par définition de l'Exemption de Motif modulo $[\![]\!]_s$, pour tout motif q, si t est exempt de q modulo $[\![]\!]_s$, on a donc bien v exempt de q modulo $[\![]\!]_s$. Enfin, si $\mathcal{R}$ est complet, comme v n'est pas réductible dans $\mathcal{R}$, v est une valeur exempte de q, *i.e.* pour toute position $\omega \in \mathcal{P}os(v)$, $q \not\preccurlyeq v_{|\omega}$. □

On observe que, bien que la méthode d'analyse linéaire ne dépendent pas nécessairement des propriétés de terminaison et de confluence du CBTRS considéré (cf. Théorème 4.18), ce n'est pas le cas de l'approche non-linéaire considérée. En effet, l'analyse non-linéaire ne permet pas de démontrer la préservation de la sémantique à chaque pas de réduction mais garantit que la préservation est éventuellement obtenue sur plusieurs pas (cf. Proposition 5.16). Ainsi, dans le cas d'une réduction non-confluente, il est possible d'avoir plusieurs dérivations d'un même terme dont l'une ne préserve pas la sémantique. De même, si la réduction d'un terme n'est pas (au moins faiblement) terminante, alors la dérivation du terme peut ne jamais préserver la sémantique du terme de départ.

Exemple 5.14. *On reprend l'algèbre de termes définie par la signature* Σ_{nl} *considérée dans l'Exemple 3.7 et on étudie le système* $\mathcal{R}$ *suivant*

$$\begin{cases} f(c(x,y)) & \to & c(x,x) \\ f(d(x)) & \to & c(g(x),g(x)) \\ g(c(a,x)) & \to & x \\ g(c(x,a)) & \to & g(c(b,x)) \\ g(c(x,x)) & \to & g(c(x,b)) \\ g(d(x)) & \to & g(x) \end{cases}$$

où les symboles définis $f^{!\mathcal{P}_f}$ *et* $g^{!\mathcal{P}_g}$ *sont annotés avec* $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}, \mathcal{P}_g = \emptyset$*.*

On peut montrer, comme on le fera pour le système considéré dans l'Exemple 5.4, que le système préserve la sémantique substitutive. Cependant ce dernier n'est ni confluent, ni fortement terminant. On peut alors s'intéresser au terme $t = f(d(c(a,a)))$ *puisque, par définition, il est exempt de* $c(a,b)$*, mais ce n'est pas le cas de toutes ses réductions :*

- *On peut observer que* $t \Longrightarrow_{\mathcal{R}} c(g(c(a,a)),g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,a)$. *On voit ici que la propriété d'Exemption de Motif est perdue au deuxième pas de réduction mais retrouvée au dernier pas. La Proposition 5.16 garantit en effet qu'il existe une telle réduction du terme* t.
- *En revanche, comme le système n'est pas confluent, il existe également des réductions du terme* t *où l'Exemption de Motif n'est pas retrouvée. Par exemple, on a :* $t \Longrightarrow_{\mathcal{R}} c(g(c(a,a)),g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(a,b))) \Longrightarrow_{\mathcal{R}} c(a,b)$.
- *De même, le système n'étant pas terminant, il peut exister des dérivations infinies de la relation de réécriture telles que la propriété d'Exemption de Motif n'est jamais retrouvée :* $t \Longrightarrow_{\mathcal{R}} c(g(c(a,a)),g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(a,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(b,a))) \Longrightarrow_{\mathcal{R}} c(a,g(c(b,b))) \Longrightarrow_{\mathcal{R}} c(a,g(c(b,b))) \Longrightarrow_{\mathcal{R}} \cdots$.

Comme présenté dans la Section précédente, la méthode d'analyse non-linéaire se présente, pour chaque règle du CBTRS considéré, en deux étapes :

1. une étape d'inférence des substitutions valeurs satisfaisant la pré-condition du profil considérée, basée sur la Proposition 5.21 ;
2. une étape de vérification de la post-condition utilisant la procédure de décision présentée dans la Figure 5.1 pour inférer des contraintes vérifiées par l'algorithme `checkInstance`.

La première étape est une étape d'inférence (quasiment) identique à celle de l'approche linéaire. Étant donné une règle $f_s^{!\mathcal{P}}(ls_1,\cdots,ls_n) \twoheadrightarrow rs$ et un profil $p_1 * \cdots * p_n \mapsto p$, on caractérise les substitutions vérifiant la pré-condition définie par le membre gauche du profil en réduisant avec $\mathfrak{R}$ le n-uplet de conjonctions $(\!|ls_1 \times z_1 s_1^{-p_1}, \ldots, ls_n \times z_n s_n^{-p_n}|\!)$. Les Propositions 5.4 et 4.10 garantissent qu'on obtient une somme de n-uplets $(\!|\lambda_1,\ldots,\lambda_n|\!)$ où chaque λ_i est une version aliassée de ls_i, et tel qu'on peut donc extraire la substitution aliassante $\sigma^{@(\!|\lambda_1,\ldots,\lambda_n|\!)}_{(\!|ls_1,\cdots,ls_n|\!)}$. Par aliasing, on doit alors montrer que la post-condition du profil est respectée, en vérifiant que $[\![\sigma^{@(\!|\lambda_1,\ldots,\lambda_n|\!)}_{(\!|ls_1,\cdots,ls_n|\!)}(rs)]\!]_s \subseteq [\![x_s^{-p}]\!]$.

Comme il n'est pas possible de construire un équivalent sémantique prenant en compte l'ensemble des contraintes de corrélation découlant de la non-linéarité du membre droit de la règle, la méthode d'analyse non-linéaire se distingue de l'approche linéaire en passant directement à l'étape de vérification. Pour cela, on construit, à l'aide de la procédure de décision présentée dans la Figure 5.1, un contexte d'évaluation explicitant des contraintes d'instanciation des variables dans le cas où la post-condition ne serait pas respectée, *i.e.* une dérivation vérifiant $[\![\sigma^{@(\!|\lambda_1,\ldots,\lambda_n|\!)}_{(\!|ls_1,\cdots,ls_n|\!)}(rs)]\!]_s \setminus [\![x_s^{-p}]\!] \neq \emptyset$. S'il n'est pas possible de construire une telle dérivation, la post-condition est donc vérifiée, et sinon, il faut vérifier, pour chaque dérivation possible, que les contraintes d'instanciations décrites par le contexte d'évaluation sont satisfaisables. L'algorithme `checkInstance` permet alors de vérifier chaque contrainte individuellement et de conclure quant à la satisfaction du profil considéré.

5.4.2 Cas d'étude

On propose d'étudier en détail le CBTRS $\mathcal{R}$ présenté dans l'Exemple 5.4. Ce dernier est défini dans l'algèbre de termes de signature $\Sigma_{nl} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{S1} & := & c(\mathsf{S2},\mathsf{S2}) & \mathsf{S2} & := & a \\ & | & d(\mathsf{S1}) & & | & b \end{array}$$

avec les symboles définis $\mathcal{D} = \{f^{!\mathcal{P}_f} : \mathsf{S1} \mapsto \mathsf{S1}, g^{!\mathcal{P}_g} : \mathsf{S1} \mapsto \mathsf{S2}, h^{!\mathcal{P}_h} : \mathsf{S2} * \mathsf{S2} \mapsto \mathsf{S1}\}$ annotés avec $\mathcal{P}_f = \{\bot \mapsto c(a,b)\}, \mathcal{P}_g = \emptyset$ et $\mathcal{P}_h = \{b * \bot \mapsto c(a,b), \bot * a \mapsto c(a,b)\}$.

On veut donc prouver que le CBTRS $\mathcal{R}$ suivant préserve la sémantique $[\![]\!]_s$:

$$\begin{cases} f(c(x,a)) & \rightarrowtriangle & c(x,x) \\ f(c(x,b)) & \rightarrowtriangle & h(x,x) \\ f(d(x)) & \rightarrowtriangle & c(g(x),g(x)) \\ g(c(x,y)) & \rightarrowtriangle & a \\ g(d(x)) & \rightarrowtriangle & b \\ h(x,y) & \rightarrowtriangle & c(y,x) \end{cases}$$

Pour les trois règle de f, on doit donc vérifier qu'elles satisfont le profil $\bot \mapsto c(a,b)$, et pour la règle de h, qu'elle satisfait les deux profils $b * \bot \mapsto c(a,b)$ et $\bot * a \mapsto c(a,b)$:

- règle $f(c(x,a)) \rightarrowtriangle c(x,x)$:

  1. On a
     - $\lambda = (c(x,a) \times z_{\mathsf{S1}}^{-\bot}) \downarrow_{\mathfrak{R}=} c(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, a)$;
     - $\sigma^{@\lambda}_{c(x,a)} = \{x \mapsto x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{c(x,a)}(c(x,x)) = c(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, x \mathbin{@} z_{1\mathsf{S2}}^{-\bot})$.

     Donc d'après la Proposition 5.19, la règle satisfait le profil $\bot \mapsto c(a,b)$ si et seulement si $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.
  2. Il existe une unique dérivation vérifiant $[\![\rho]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \mapsto a \setminus \bot \vDash [\![x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{A}_\times) \qquad \dfrac{}{x \mapsto b \setminus \bot \vDash [\![x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{x \mapsto \bot \setminus \bot \vDash [\![x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}]\!]_s \cap [\![a]\!] \neq \emptyset \wedge [\![x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![b]\!] \neq \emptyset}\,(\wedge)}{x \mapsto \bot \setminus \bot \vDash [\![c(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, x \mathbin{@} z_{1\mathsf{S2}}^{-\bot})]\!]_s \cap [\![c(a,b)]\!] \neq \emptyset}\,(\mathbf{C}_\times^{=})}{x \mapsto \bot \setminus \bot \vDash [\![c(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, x \mathbin{@} z_{1\mathsf{S2}}^{-\bot})]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}\,(\mathbf{C}_\setminus^{\not<})$$

     Et on a clairement $[\![\bot]\!] \setminus [\![\bot]\!] = \emptyset$, donc d'après la Proposition 5.23, on a $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$. Donc la règle satisfait bien le profil $\bot \mapsto c(a,b)$.

- règle $f(c(x,b)) \rightarrowtriangle h(x,x)$:

  1. On a
     - $\lambda = (c(x,b) \times z_{\mathsf{S1}}^{-\bot}) \downarrow_{\mathfrak{R}=} c(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, b)$;
     - $\sigma^{@\lambda}_{c(x,b)} = \{x \mapsto x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}\}$;
     - d'où $\rho = \sigma^{@\lambda}_{c(x,b)}(h(x,x)) = h(x \mathbin{@} z_{1\mathsf{S2}}^{-\bot}, x \mathbin{@} z_{1\mathsf{S2}}^{-\bot})$.

     Donc d'après la Proposition 5.19, la règle satisfait le profil $\bot \mapsto c(a,b)$ si et seulement si $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.
  2. On a montré dans l'Exemple 5.12 que $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$. Donc la règle satisfait bien le profil $\bot \mapsto c(a,b)$.

- règle $f(d(x)) \rightarrowtriangle c(g(x),g(x))$

1. On a
   - $\lambda = (d(x) \times y_{\mathsf{S1}}^{-\perp}) \downarrow_{\mathfrak{R}} = d(x \mathbin{@} z_{\mathsf{S1}}^{-\perp})$;
   - $\sigma_{d(x)}^{@\lambda} = \{x \mapsto x \mathbin{@} z_{\mathsf{S1}}^{-\perp}\}$;
   - d'où $\rho = \sigma_{d(x)}^{@\lambda}(c(g(x), g(x))) = c(g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}), g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}))$.

   Donc d'après la Proposition 5.19, la règle satisfait le profil $\perp \mapsto c(a, b)$ si et seulement si $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.
2. Il existe une unique dérivation vérifiant $[\![\rho]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y \mapsto a \setminus \perp \vDash [\![y \mathbin{@} z_{1_{\mathsf{S2}}}^{-\perp}]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{g(x) \mapsto a \setminus \perp \vDash [\![g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp})]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{F}_\times) \quad \dfrac{\dfrac{}{y \mapsto b \setminus \perp \vDash [\![y \mathbin{@} z_{1_{\mathsf{S2}}}^{-\perp}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{g(x) \mapsto b \setminus \perp \vDash [\![g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp})]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{F}_\times)}{g(x) \mapsto \perp \setminus \perp \vDash [\![g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp})]\!]_s \cap [\![a]\!] \neq \emptyset \wedge [\![g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp})]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\wedge)}{g(x) \mapsto \perp \setminus \perp \vDash [\![c(g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}), g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}))]\!]_s \cap [\![c(a, b)]\!] \neq \emptyset}\,(\mathbf{C}_\times^{=})}{g(x) \mapsto \perp \setminus \perp \vDash [\![c(g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}), g(x \mathbin{@} z_{\mathsf{S1}}^{-\perp}))]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}\,(\mathbf{C}_{\setminus}^{\not=})$$

   Et on a clairement $[\![\perp]\!] \setminus [\![\perp]\!] = \emptyset$, donc d'après la Proposition 5.23, on a $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$. Donc la règle satisfait bien le profil $\perp \mapsto c(a, b)$.

- règle $h(x, y) \twoheadrightarrow c(y, x)$ et le profil $b * \perp \mapsto c(a, b)$:

1. On a
   - $\lambda = (\!|x \times z_{1_{\mathsf{S2}}}^{-b}, y \times z_{2_{\mathsf{S2}}}^{-\perp}|\!) \downarrow_{\mathfrak{R}} = (\!|x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b}, y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}|\!)$;
   - $\sigma_{(\!|x,y|\!)}^{@\lambda} = \{x \mapsto x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b}, y \mapsto y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}\}$;
   - d'où $\rho = \sigma_{c(x,a)}^{@\lambda}(c(y, x)) = c(y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}, x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b})$.

   Donc d'après la Proposition 5.19, la règle satisfait le profil $b * \perp \mapsto c(a, b)$ si et seulement si $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.
2. Il existe une unique dérivation vérifiant $[\![\rho]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y \mapsto a \setminus \perp \vDash [\![y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{A}_\times) \quad \dfrac{}{x \mapsto \perp \setminus \perp \vDash [\![x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{y \mapsto a \setminus \perp; x \mapsto \perp \setminus \perp \vDash [\![y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}]\!]_s \cap [\![a]\!] \neq \emptyset \wedge [\![x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b}]\!]_s \setminus [\![b]\!] \neq \emptyset}\,(\wedge)}{y \mapsto a \setminus \perp; x \mapsto \perp \setminus \perp \vDash [\![c(y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}, x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b})]\!]_s \cap [\![c(a, b)]\!] \neq \emptyset}\,(\mathbf{C}_\times^{=})}{y \mapsto a \setminus \perp; x \mapsto \perp \setminus \perp \vDash [\![c(y \mathbin{@} z_{2_{\mathsf{S2}}}^{-\perp}, x \mathbin{@} z_{1_{\mathsf{S2}}}^{-b})]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}\,(\mathbf{C}_{\setminus}^{\not=})$$

   Et on a clairement $[\![\perp]\!] \setminus [\![\perp]\!] = \emptyset$, donc d'après la Proposition 5.23, on a $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$. Donc la règle satisfait bien le profil $b * \perp \mapsto c(a, b)$.

- règle $h(x, y) \twoheadrightarrow c(y, x)$ et le profil $\perp * a \mapsto c(a, b)$:

1. On a
   - $\lambda = (\!|x \times z_{1_{\mathsf{S2}}}^{-\perp}, y \times z_{2_{\mathsf{S2}}}^{-a}|\!) \downarrow_{\mathfrak{R}} = (\!|x \mathbin{@} z_{1_{\mathsf{S2}}}^{-\perp}, y \mathbin{@} z_{2_{\mathsf{S2}}}^{-a}|\!)$;

- $\sigma^{@\lambda}_{(\!|x,y|\!)} = \{x \mapsto x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}, y \mapsto y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}\}$;
- d'où $\rho = \sigma^{@\lambda}_{c(x,a)}(c(y,x)) = c(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-bot})$.

Donc d'après la Proposition 5.19, la règle satisfait le profil $\bot * a \mapsto c(a,b)$ si et seulement si $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$.

2. Il existe une unique dérivation vérifiant $[\![\rho]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset$:

$$\dfrac{\dfrac{\dfrac{}{y \mapsto \bot \setminus \bot \vDash [\![y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{A}_\times) \qquad \dfrac{}{x \mapsto b \setminus \bot \vDash [\![x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{y \mapsto \bot \setminus \bot; x \mapsto b \setminus \bot \vDash [\![y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}]\!]_s \cap [\![a]\!] \neq \emptyset \wedge [\![x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}]\!]_s \setminus [\![b]\!] \neq \emptyset}\,(\wedge)}{y \mapsto \bot \setminus \bot; x \mapsto b \setminus \bot \vDash [\![c(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot})]\!]_s \cap [\![c(a,b)]\!] \neq \emptyset}\,(\mathbf{C}^{=}_{\times})}{y \mapsto \bot \setminus \bot; x \mapsto b \setminus \bot \vDash [\![c(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-a}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot})]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}\,(\mathbf{C}^{\not=}_{\setminus})$$

Et on a clairement $[\![\bot]\!] \setminus [\![\bot]\!] = \emptyset$, donc d'après la Proposition 5.23, on a $[\![\rho]\!]_s \subseteq [\![x_{\mathsf{S1}}^{-c(a,b)}]\!]$. Donc la règle satisfait bien le profil $\bot * a \mapsto c(a,b)$.

On en déduit que le système $\mathcal{R}$ préserve la sémantique $[\![]\!]_s$. Le système étant de plus confluent et terminant, la Proposition 5.16 garantit que pour tout $t \in \mathcal{T}(\mathcal{F})$, on a $(t) \downarrow_{\mathcal{R}} \in [\![t]\!]_s$.

5.4.3 Limitations de l'analyse non-linéaire

Comme la méthode d'analyse se repose sur la notion de sémantique substitutive (Définition 5.7), elle hérite naturellement de ses limitations. En effet, cette notion est définie de façon à prendre en compte toutes les contraintes de corrélation entre différentes instances d'une même variable ou différentes occurrences d'un même sous-terme, mais ne prend pas en compte des contraintes similaires qui résulteraient de l'instanciation d'une variable et/ou de la réduction d'un sous-terme.

On avait ainsi vu dans l'Exemple 5.7, que le terme $c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(x))$, où $id^{!\mathcal{P}_{id}}$ est annoté avec $\mathcal{P}_{id} = \{b \mapsto b\}$, n'est pas exempt de $c(a,b)$ modulo $[\![]\!]_s$. Cependant, comme x n'a que deux instanciations possibles dans la signature Σ_{nl}, on peut vérifier que pour chaque instanciation, le terme est bien exempt de $c(a,b)$:

- Pour x instancié par a, le terme $c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(a))$ est exempt de $c(a,b)$ modulo $[\![]\!]_s$ parce que $id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(a)$ est exempt de b.
- Pour x instancié par b, le terme $c(id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b), id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b))$ est exempt de $c(a,b)$ modulo $[\![]\!]_s$ par contrainte de corrélation entre les deux occurrences de $id_{\mathsf{S2}}^{!\mathcal{P}_{id}}(b)$.

De même, en considérant un symbole défini $\varphi^{!\mathcal{P}} : \mathsf{S2} * \mathsf{S2} \mapsto \mathsf{S2}$ annoté avec $\mathcal{P} = \{b * \bot \mapsto b, \bot * b \mapsto b\}$, le terme $c(\varphi(x,y), \varphi(y,x))$ n'est pas exempt de $c(a,b)$ modulo $[\![]\!]_s$ puisque la sémantique ne fait aucune corrélation entre les deux sous-termes $\varphi(x,y)$ et $\varphi(y,x)$ (bien qu'elle considère correctement les contraintes de corrélations entre les différentes instances des variables x et y). On peut le voir, notamment grâce à la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z \mapsto a \setminus \bot \vDash [\![z \mathbin{@} z_3{}_{\mathsf{S2}}^{-\bot}]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{\varphi(x,y) \mapsto a \setminus \bot \vDash [\![\varphi(x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}, y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot})]\!]_s \cap [\![a]\!] \neq \emptyset}\,(\mathbf{F}_\times) \qquad \dfrac{\vdots}{\Gamma \vDash [\![\varphi(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}, y \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot})]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{F}_\times)}{\Gamma; \varphi(x,y) \mapsto a \setminus \bot \vDash [\![\varphi(x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}, y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot})]\!]_s \cap [\![a]\!] \neq \emptyset \wedge [\![\varphi(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot})]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\wedge)}{\Gamma; \varphi(x,y) \mapsto a \setminus \bot \vDash [\![c(\varphi(x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}, y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}), \varphi(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}))]\!]_s \cap [\![c(a,b)]\!] \neq \emptyset}\,(\mathbf{C}^{=}_{\times})}{\Gamma; \varphi(x,y) \mapsto a \setminus \bot \vDash [\![c(\varphi(x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}, y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}), \varphi(y \mathbin{@} z_2{}_{\mathsf{S2}}^{-\bot}, x \mathbin{@} z_1{}_{\mathsf{S2}}^{-\bot}))]\!]_s \setminus [\![x_{\mathsf{S1}}^{-c(a,b)}]\!] \neq \emptyset}\,(\mathbf{C}^{\not=}_{\setminus})$$

avec $\Gamma = \{y \mapsto z_2{}^{-\bot}_{\mathsf{S2}} \setminus x^{-b}_{\mathsf{S2}}; x \mapsto z_1{}^{-\bot}_{\mathsf{S2}} \setminus x^{-b}_{\mathsf{S2}}; \varphi(y,x) \mapsto b \setminus \bot\}$, et la branche droite de la dérivation :

$$\dfrac{\dfrac{\dfrac{}{\Gamma_y \vDash [\![y \mathbin{@} z_2{}^{-\bot}_{\mathsf{S2}}]\!]_s \setminus [\![x^{-b}_{\mathsf{S2}}]\!] \neq \emptyset}\,(\mathbf{A}_\setminus) \quad \dfrac{}{\Gamma_x \vDash [\![x \mathbin{@} z_1{}^{-\bot}_{\mathsf{S2}}]\!]_s \setminus [\![x^{-b}_{\mathsf{S2}}]\!] \neq \emptyset}\,(\mathbf{A}_\setminus) \quad \dfrac{}{z \mapsto b \setminus \bot \vDash [\![z \mathbin{@} z_3{}^{-\bot}_{\mathsf{S2}}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{A}_\times)}{\Gamma_y; \Gamma_x; z \mapsto b \setminus \bot \vDash [\![y \mathbin{@} z_2{}^{-\bot}_{\mathsf{S2}}]\!]_s \setminus [\![x^{-b}_{\mathsf{S2}}]\!] \neq \emptyset \wedge [\![x \mathbin{@} z_1{}^{-\bot}_{\mathsf{S2}}]\!]_s \setminus [\![x^{-b}_{\mathsf{S2}}]\!] \neq \emptyset \wedge [\![z \mathbin{@} z_3{}^{-\bot}_{\mathsf{S2}}]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\wedge)}{\Gamma \vDash [\![\varphi(y \mathbin{@} z_2{}^{-\bot}_{\mathsf{S2}}, y \mathbin{@} z_1{}^{-\bot}_{\mathsf{S2}})]\!]_s \cap [\![b]\!] \neq \emptyset}\,(\mathbf{F}_\times)$$

avec $\Gamma_x = \{x \mapsto z_1{}^{-\bot}_{\mathsf{S2}} \setminus x^{-b}_{\mathsf{S2}}\}$ et $\Gamma_y = \{y \mapsto z_2{}^{-\bot}_{\mathsf{S2}} \setminus x^{-b}_{\mathsf{S2}}\}$.

Cependant, comme pour le cas précédent, les variables x et y n'ont chacune que deux instanciations possibles dans la signature Σ_{nl}. Donc il n'y a que quatre instanciations différentes des deux variables, dont deux où les instanciations de $\varphi(x,y)$ et $\varphi(y,x)$ sont identiques. Pour ces deux cas, la contrainte de corrélation entre ces deux occurrences identiques garantit l'Exemption du Motif $c(a,b)$. On peut vérifier que pour les instanciations restantes c'est également le cas :

- Pour x instancié par a et y par b, le terme $c(\varphi^{!\mathcal{P}}(a,b), \varphi^{!\mathcal{P}}_{\mathsf{S2}}(b,a))$ est exempt de $c(a,b)$ modulo $[\![]\!]_s$, car $\varphi^{!\mathcal{P}}(b,a)$ est exempt de b.
- Pour x instancié par b et y par a, le terme $c(\varphi^{!\mathcal{P}}(b,a), \varphi^{!\mathcal{P}}_{\mathsf{S2}}(a,b))$ est exempt de $c(a,b)$ modulo $[\![]\!]_s$, car $\varphi^{!\mathcal{P}}(a,b)$ est exempt de b.

Malgré tout, l'analyse basée sur la sémantique $[\![]\!]_s$ ne permet donc pas de vérifier ces propriétés. La méthode d'analyse pourrait donc échouer à vérifier la préservation de la sémantique par une relation de réécriture induite par un CBTRS possédant des règles ayant de tels termes en membres droits.

On parlera plus en détails des limitations de l'approche par Exemption de Motif (linéaire ou non), dans le Chapitre suivant, en la comparant à d'autres approches présentées dans le Chapitre 2.

5.5 Synthèse

Dans ce Chapitre, nous avons étudié l'application de la méthode d'analyse par Exemption de Motif, introduite dans le Chapitre précédent, aux CBTRS non-linéaires. Cette analyse ayant pour but de vérifier que les spécifications données par le système d'annotation choisi sont en accord avec le CBTRS décrivant le comportement des fonctions associées aux symboles annotés, elle repose sur l'équivalence entre la préservation de la sémantique close (Définition 3.14) et la satisfaction de profil (Définition 3.15), établie par la Proposition 3.18. Cependant, dans le cas de termes non-linéaires, la sémantique close n'étant pas forcément préservée par substitution, l'approche introduite ne s'applique que sous certaines restrictions. Pour pouvoir appliquer la méthode d'analyse, on a donc présenté deux approches :

- une approche par linéarisation qui consiste à étudier un système non-linéaire en sur-approximant la sémantique de ses membres droits en les linéarisant ;
- une approche stricte qui, sous hypothèse d'une stratégie de réduction stricte, permet de prendre en compte certaines contraintes de corrélation entre plusieurs instances d'une même variable en se basant sur la méthode d'aliasing utilisée pour l'inférence de contraintes sur les variables.

Ces deux approches présentent néanmoins certaines limitations, dûes notamment à l'utilisation de la sémantique close. Avec une hypothèse de confluence (qui n'est généralement pas contraignante pour l'étude de programmes fonctionnels), on a donc étudié plusieurs formes de sémantique (Définition 5.7) permettant une approximation plus précise de l'ensemble des formes

normale potentielles des termes. La sémantique finalement retenue utilise une approche par substitution qui permet de considérer toutes les contraintes de corrélation du terme d'origine en omettant celles issues de l'instanciation effective du terme. De plus, comme pour la notion de sémantique close, cette sémantique substitutive supporte des notions de préservation par réécriture (Définition 5.8) et de satisfaction de profil (Définition 5.9), telles que la sémantique substitutive est préservée par une règle de réécriture si et seulement cette dernière satisfait tous les profils de l'annotation du symbole de tête de son membre gauche.

Comme dans le cas linéaire, on propose donc une méthode d'analyse statique vérifiant que chaque règle du CBTRS considéré satisfait par substitution tous les profils de l'annotation du symbole de tête de son membre gauche. Cette méthode fonctionne de façon similaire à celle proposée dans le Chapitre précédent et réutilise la plupart des mécanismes mis en œuvre par cette dernière. Elle se présente en effet en deux étapes :

1. une étape d'inférence des substitutions valeurs satisfaisant la pré-condition du profil considérée, basée sur la Proposition 5.21 ;
2. une étape de vérification de la post-condition utilisant la procédure de décision présentée dans la Figure 5.1 pour inférer des contraintes vérifiée par l'algorithme `checkInstance`.

Pour la vérification de la post-condition, on n'utilise plus d'équivalent sémantique mais on construit, à l'aide de la procédure de décision présentée dans la Figure 5.1, un contexte d'évaluation explicitant des contraintes d'instanciation des variables dans le cas où la post-condition ne serait pas respectée. Si ces contraintes ne sont pas vérifiables, on peut alors conclure que la règle satisfait le profil considéré.

Dans le dernier Chapitre, on propose d'étudier la complexité des méthodes linéaires et non-linéaires, et de présenter notre implémentation de ces méthodes d'analyse. On présentera notamment les résultats obtenus par notre implémentation, que l'on pourra ainsi comparer aux approches présentées dans la Section 2.3.

6

Implémentation : Optimisations et résultats

La méthode d'analyse présentée dans le Chapitre 4 et son adaptation pour les systèmes non-linéaires (cf. Chapitre 5) ont été implémentée en Haskell. Le code source, ainsi qu'un jeu de fichiers de test comprenant la plupart des exemples considérés dans ce mémoire et ceux présentées dans l'Annexe C, sont disponibles sur le dépôt [CL20] [1].

L'implémentation prend donc en entrée un fichier définissant la signature de l'algèbre considérée, les annotations des symboles définis et un CBTRS, et vérifie que le système préserve la sémantique. Dans le cas contraire, on renvoie une liste des règles ne préservant pas la sémantique. De plus, l'implémentation propose une branche dédiée aux tests de performance, permettant d'analyser les performances de la méthode proposée sur les exemples fournis, ainsi que sur des cas de taille donnée, générés aléatoirement.

On présente dans ce Chapitre les grandes lignes de cette implémentation, et on discute de la complexité de la méthode d'analyse, que l'on compare aux résultats expérimentaux obtenus.

Dans un premier temps, on discute de la complexité des méthodes d'analyse linéaire et non-linéaire, et on présente les implémentations respectives. Dans la Section 6.2, on propose certaines optimisations considérées dans l'implémentation. Et enfin, dans la Section 6.3, on présente les résultats expérimentaux obtenus que l'on compare avec l'état de l'art.

L'ensemble des tests mentionnés dans ce Chapitre ont été effectués sur un ordinateur équipé d'un processeur Intel Core I5-8250U avec 8Go de mémoire vive.

6.1 Implémentation

On présente l'implémentation de la méthode d'analyse proposée pour les CBTRS linéaires, puis son adaptation pour les CBTRS non-linéaires. On présente également des ordres de grandeur de complexité des méthodes par rapport à certains paramètres.

6.1.1 Cas Linéaire

Pour l'analyse statique de CBTRS linéaires, on se repose sur la notion de satisfaction de profil (Définition 3.15) pour vérifier que la relation de réécriture induite préserve la sémantique.

1. une version testable en ligne de l'implémentation est également disponible sur `http://htmlpreview.github.io/?https://github.com/plermusiaux/pfree_check/blob/webnix/out/index.html`

Pour cela, la Proposition 3.18 établit qu'il est nécessaire et suffisant de montrer que chaque règle satisfait tous les profils du symbole de tête du membre gauche.

Pour chaque règle et chaque profil de l'annotation du symbole défini en tête de son membre gauche, la méthode d'analyse se présente en trois étapes :

1. une étape d'inférence des substitutions considérées par la notion de satisfaction de profil basée sur la Proposition 4.15 ;
2. une étape de construction de l'équivalent sémantique (Définition 4.7) pour ramener les termes inférés à des motifs constructeurs ;
3. une étape de calcul des sémantiques pour prouver que l'intersection de sémantiques considérée par la Proposition 3.9 est bien vide.

Pour chaque règle, on construit ainsi une forme inférée de la règle satisfaisant la pré-condition donnée par le membre gauche de chaque profil. La méthode d'analyse teste alors, pour chaque règle inférée ainsi obtenue, que la post-condition, donnée par le membre droit du profil, est vérifiée. Pour toute règle inférée ne vérifiant pas sa post-condition, l'implémentation renvoie alors la liste des termes de la décomposition de la sémantique profonde du membre droit dont la conjonction avec le motif à droite du profil a une sémantique non-vide.

Les trois étapes se reposent majoritairement sur l'algorithme `getReachable`, présenté en Figure 4.7, et le système $\mathfrak{R}$, présenté en Figure 4.8. Tandis que l'implémentation reprend de façon quasi-exacte la logique du système $\mathfrak{R}$, on a fait le choix d'une implémentation moins littérale de l'algorithme `getReachable`.

Plutôt que la version initiale de l'algorithme, l'implémentation propose d'utiliser celle présentée en Figure 6.1. L'idée générale de cette version repose sur le même modèle de graphe d'atteignabilité (cf. Définition 4.3) pour fournir une décomposition de la sémantique profonde d'une (quasi-)variable annotée, comme garanti par le Théorème 4.5. Cependant, là où l'algorithme original proposait d'obtenir la décomposition en construisant, dans un premier temps, le graphe total (cf. Définition 4.1), pour en déduire ensuite le graphe d'atteignabilité, avant de conclure en extrayant les nœuds atteignables depuis le nœud d'origine considéré, la version utilisée tente d'effectuer l'ensemble de ces étapes simultanément.

Cette approche donne, en théorie, une complexité pire cas plus élevée puisqu'elle ne retient pas les nœuds déjà explorés quand leur sémantique est vide, ou qu'ils ne sont atteignables que via un nœud dont la sémantique est vide [1]. En pratique, l'algorithme permet une amélioration du temps de calcul de l'algorithme `getReachable` d'environ 55%, pour des entrées avec des symboles constructeurs d'arité 6, et des motifs annotants contenant 100 symboles constructeurs. A l'échelle de la méthode d'analyse, cela se traduit par une réduction du temps de calcul de 61% en moyenne sur l'analyse d'une règle avec un terme droit contenant 100 symboles constructeurs et des annotations contenant 50 symboles constructeurs. En moyenne, sur les scénarios fournis avec l'implémentation (et présentés dans l'Annexe C), on observe un gain de temps moyen de 44%. Un tableau de performance détaillant ces résultats est présenté en Figure 6.2. [2]

Outre l'amélioration des performances, un intérêt majeur de cette approche est qu'elle permet de prendre en considération des termes infinis de la (co-)algèbre considérée, obtenus en considérant une propriété d'instanciabilité circulaire des nœuds du graphe. En effet, dans le contexte de programmes écrits dans des langages proposant une stratégie d'évaluation *lazy* (comme Haskell, par exemple), il est courant de se reposer sur cette stratégie d'évaluation pour travailler avec des termes potentiellement infinis, sans nécessiter leur évaluation exacte.

1. certains ajustements permettant de limiter cet effet ont été considérés, mais aucune amélioration de performance n'a été observée sur les tests

2. Un gain de temps similaire est observé, en utilisant une analyse stricte, comme présentée dans le Chapitre 5

Fonction `getReachable`(s, p, S, r)
- **Entrées :** s : sorte,
  p : motif annotant,
  S : ensemble de nœuds atteints (initialisé à $\emptyset$),
  r : motif complément
- **Résultat :** ensemble de nœuds variables instanciables et atteignables depuis $x_s^{-p} \setminus r$
- **si** $p : s$ **alors** $r \longleftarrow r + p$
- **si** $[\![x_s \setminus r]\!] = \emptyset$ **alors retourner** $\emptyset$
- **si** $\exists (s, r') \in S, [\![r']\!] = [\![r]\!]$ **alors retourner** S
- $R \longleftarrow S \cup \{(s, r)\}$
- $reachable \longleftarrow False$
- **pour** $c \in \mathcal{C}_s$ **faire**
  - $Q_c \longleftarrow \{(\overbrace{\bot, \ldots, \bot}^{m})\}$ avec $m = arity(c)$
  - **pour** $i = 1$ **à** n *avec* $r = \sum_{i=1}^{n} r_i$ **faire**
    - **si** $r_i(\epsilon) = c$ **alors**
      - $tQ_c \longleftarrow \emptyset$
      - **pour** $(q_1, \ldots, q_m) \in Q_c, k \in [1, m]$ **faire**
        - $tQc \longleftarrow \{(q_1, \ldots, q_k + r_{i|k}, \ldots, q_m)\} \cup tQc$
      - $Q_c \longleftarrow tQ_c$
  - **pour** $(q_1, \ldots, q_m) \in Q_c$ **faire**
    - $R' \longleftarrow R$
    - $i \longleftarrow 0$ **tant que** $R' \neq \emptyset \wedge i < m$ **faire**
      - $i \longleftarrow i + 1$
      - $R' \longleftarrow$ `getReachable`$(\mathcal{D}om(c)[i], p, R', q_i)$
    - **si** $R' \neq \emptyset$ **alors**
      - $reachable \longleftarrow True$
      - $R \longleftarrow R'$
- **si** $reachable$ **alors retourner** R **sinon retourner** $\emptyset$

FIGURE 6.1 – Algorithme `getReachable` utilisé dans l'implémentation

Exemple 6.1. *On considère l'algèbre de termes définie par la signature* Σ_{list} *introduite dans l'Exemple 3.5, avec les symboles définis* $\mathcal{D} = \{repeat : \mathsf{Expr} \mapsto \mathsf{List}\}$ *où* $repeat^{!\mathcal{P}}$ *est annoté avec* $\mathcal{P} = \{nil \mapsto cons(lst(l_1), l_2) + nil\}$*. Et on vérifie que le système* $\mathcal{R}$ *suivant préserve la sémantique :*

$$\{\ repeat(e) \quad \rightarrowtriangle \quad cons(e, repeat(e))$$

La fonction associée au symbole repeat est très couramment utilisée dans les langages fonctionnels lazy *pour générer des listes de taille arbitraire. Ici le profil considéré prétend que pour toute entrée n'étant pas une liste finie, le résultat doit être une liste plate. On peut cependant observer que toute réduction du terme* $repeat(lst(repeat(\mathsf{Int}(z))))$ *n'est pas une liste plate.*

La méthode d'analyse proposée dans le Chapitre 4 permet néanmoins de vérifier la satisfaction du profil[1] *:*

1. *On a*
   - $(e \times x_{\mathsf{Expr}}^{-nil}) \downarrow_{\mathfrak{R}} = e \mathbin{@} x_{\mathsf{Expr}}^{-nil}$

1. le système $\mathcal{R}$ est en fait non-linéaire, mais le résultat serait le même avec les méthodes non-linéaires.

	analyse linéaire Fig 6.1	analyse linéaire Fig 4.7	analyse stricte Fig 6.1	analyse stricte Fig 4.7
flatten1	$23\mu s$	$29\mu s$	$23\mu s$	$29\mu s$
flatten2	$41\mu s$	$52\mu s$	$42\mu s$	$52\mu s$
flatten3	$37\mu s$	$43\mu s$	$35\mu s$	$41\mu s$
flatten fail	$31\mu s$	$41\mu s$	$28\mu s$	$38\mu s$
negativeNF	$387\mu s$	$688\mu s$	$359\mu s$	$659\mu s$
skolem	$47\mu s$	$71\mu s$	$47\mu s$	$70\mu s$
removePlus0 fail	$26\mu s$	$37\mu s$	$23\mu s$	$35\mu s$
removePlus0	$50\mu s$	$93\mu s$	$50\mu s$	$93\mu s$
multiply0	$11\mu s$	$12\mu s$	$12\mu s$	$13\mu s$
insertionSort*	$201\mu s$	$259\mu s$	$195\mu s$	$253\mu s$
mergeSort*	$847\mu s$	$992\mu s$	$569\mu s$	$714\mu s$
reverse	$125\mu s$	$226\mu s$	$128\mu s$	$229\mu s$
reverse2	$360\mu s$	$671\mu s$	$370\mu s$	$686\mu s$
delete*	$113\mu s$	$133\mu s$	$108\mu s$	$130\mu s$
sortedDelete*	$265\mu s$	$359\mu s$	$228\mu s$	$322\mu s$
otrs	$1.40ms$	$2.48ms$	$1.40ms$	$2.48ms$
TRS aléatoire	$486ms$	$734ms$	$542ms$	$789ms$
terme aléatoire	$40ms$	$144ms$	$47ms$	$151ms$

FIGURE 6.2 – Différences de performance en temps d'exécution entre l'algorithme `getReachable` théorique et implémenté ; les scénarios *TRS aléatoire* et *terme alétoire* correspondent à la vérification d'un TRS de 25 règles et d'une propriété d'Exemption de Motif d'un terme ; les autres scénarios sont présentés en Appendix C, avec ceux marqués * modifiés pour pouvoir être validés avec l'analyse linéaire.

- $\sigma_e^{@e@x_{\mathsf{Expr}}^{-nil}} = \{e \mapsto e \mathbin{@} x_{\mathsf{Expr}}^{-nil}\}$
- *d'où* $\rho = \sigma_e^{@e@x_{\mathsf{Expr}}^{-nil}}(cons(e, repeat(e))) = cons(e \mathbin{@} x_{\mathsf{Expr}}^{-nil}, repeat(e \mathbin{@} x_{\mathsf{Expr}}^{-nil}))$

Donc d'après la Proposition 4.15, la règle satisfait le profil si et seulement si $\{\!|\rho|\!\} \cap [\![p_{flat} + nil]\!] = \emptyset$.

2. *On calcule l'équivalent sémantique de* ρ *:*
   - *Étant donné que le seul profil du symbole repeat est* $nil \mapsto p_{flat} + nil$*, on veut vérifier si* $\{\!|e \mathbin{@} x_{\mathsf{Expr}}^{-nil}|\!\} \cap [\![nil]\!] = \emptyset$.
   - *Avec l'algorithme* **`getReachable`***, on a* $\{\!|e@x_{\mathsf{Expr}}^{-nil}|\!\} = [\![x_{\mathsf{Expr}}^{-nil}]\!] \cup [\![i_{\mathsf{Int}}^{-nil}]\!]$*, d'où* $\{\!|e@x_{\mathsf{Expr}}^{-nil}|\!\} \cap [\![nil]\!] = \emptyset$.
   - *On a donc* $\tilde{\rho} = cons(e \mathbin{@} x_{\mathsf{Expr}}^{-nil}, l \mathbin{@} z_{\mathsf{List}}^{-p_{flat}+nil})$.
3. *Or avec l'algorithme* **`getReachable`***, on a* $\{\!|l_{\mathsf{List}}^{-p_{flat}+nil}|\!\} = \emptyset$*, donc d'après la Proposition 4.1, on a* $\{\!|\tilde{\rho}|\!\} = \emptyset$*. Par conséquent,* $\{\!|\rho|\!\} \cap [\![p_{flat} + nil]\!] = \emptyset$*, donc la règle satisfait bien le profil.*

Ce résultat est parfaitement correct dans une algèbre de termes classique : il n'existe pas de valeur de sorte List *ne contenant pas le constructeur nil. La sémantique de repeat*(v)*, avec* v *un terme clos de sorte* Expr *exempt de nil, et de toutes ses réductions par* $\mathfrak{R}$ *sont donc vides. Si on considère cependant une co-algèbre avec la même définition, on peut alors prendre en compte des termes infinis, et il existe alors bien des valeurs de sorte* List *ne contenant pas le constructeur nil (i.e. une liste infinie).*

L'algorithme proposée en Figure 6.1 accepte la boucle entre le nœud l^{-nil}_{List} *et le nœud* $cons(e^{-nil}_{\mathsf{Expr}}, l^{-nil}_{\mathsf{List}})$ *comme preuve de l'existence d'un tel terme infini. On a alors :*

$$\begin{aligned}\{\!\{\tilde{\rho}\}\!\} \quad = \quad & [\![\tilde{\rho}]\!] \cup [\![e^{-nil}_{\mathsf{Expr}}]\!] \cup [\![l^{-nil}_{\mathsf{List}}]\!] \cup [\![i^{-nil}_{\mathsf{Int}}]\!] \\ & \cup [\![l^{-p_{flat}+nil}_{\mathsf{List}}]\!] \cup [\![e^{-p_{flat}+nil}_{\mathsf{Expr}} \setminus lst(l_1)]\!] \cup [\![i^{-nil}_{\mathsf{Int}}]\!]\end{aligned}$$

Et on a :

- $(\tilde{\rho} \times (p_{flat} + nil)) \downarrow_{\mathfrak{R}} = cons(e \mathbin{@} (lst(l_1 \mathbin{@} y^{-nil}_{\mathsf{List}})), l \mathbin{@} z^{-p_{flat}+nil}_{\mathsf{List}})$
- $(l^{-nil}_{\mathsf{List}} \times (p_{flat} + nil)) \downarrow_{\mathfrak{R}} = cons(e \mathbin{@} (lst(l_1 \mathbin{@} y^{-nil}_{\mathsf{List}})), l \mathbin{@} z^{-nil}_{\mathsf{List}})$

La règle ne satisfait donc plus le profil. On peut en revanche vérifier que les profils $\mathcal{P} = \{nil \mapsto nil, lst(l) \mapsto p_{flat} + nil\}$ *sont satisfaits par la règle, ce qui permet ainsi de garantir que les termes atteignables par réduction depuis un terme de la forme* $repeat(int(n))$ *sont tous exempts de* p_{flat} *et nil : une liste répétant infiniment une expression n'étant pas une liste est bien plate et infinie.*

Enfin, cette approche de calcul du graphe d'atteignabilité, permet également de stopper le calcul prématurément quand il n'est pas nécessaire de calculer l'ensemble des nœuds instanciables, et que l'on cherche simplement à vérifier que le nœud d'origine est instanciable (ce qui est notamment le cas pour l'application des règles de $\mathfrak{R}$).

De plus, en terme de complexité, la méthode d'analyse propose des performances théoriques très intéressantes, notamment par rapport aux nombres de règles et de profils :

- **[nombre de règles du CBTRS]** En termes de complexité, l'intérêt principal de la méthode d'analyse provient du fait qu'elle vérifie la satisfaction des profils de chaque règle indépendamment du reste du système. La méthode d'analyse opère ainsi avec une complexité linéaire par rapport au nombre de règles du système considéré. En pratique, on verra dans la Section suivante, que cette approche est d'ailleurs assez redondante, et on proposera une approche ayant pour objectif de limiter le nombre de calculs redondants, ce qui permet d'améliorer encore les performances de la méthode par rapport au nombre de règles.
- **[nombre de profils des annotations]** Le nombre de profils par annotation des symboles définis a un impact direct sur la complexité globale de la méthode d'analyse, puisque cette dernière vérifie que chaque règle satisfait tous les profils de l'annotation du symbole de tête à gauche de la règle. De plus le calcul d'équivalent sémantique a une complexité linéaire avec ce même nombre de profils, puisqu'il vérifie pour chaque profil de l'annotation s'il s'applique ou non au terme considéré. La méthode d'analyse a donc une complexité quadratique avec le nombre moyen de profils par annotation des symboles définis.

Ces paramètres sont particulièrement intéressants puisque le nombre de règles est directement lié à la taille du programme fonctionnel considéré, et le nombre de profils renvoie à la complexité des spécifications algébriques considérées. Cependant, la complexité exacte de la méthode d'analyse dépend également de très nombreux paramètres allant du nombre et de l'arité des symboles constructeurs de la signature, à la taille des termes et des motifs considérés.

Plusieurs mesures peuvent être considérées pour évaluer la taille d'un terme : le nombre de symboles, la profondeur du terme et l'arité des symboles. Le nombre de symboles fournissant en général une mesure plus précise de la taille concrète des termes, on s'intéresse à son impact sur la complexité de la méthode d'analyse (on discutera également plus tard de l'impact de l'arité des symboles). La même observation est également vraie pour la taille des motifs en annotation. De plus comme ces derniers mettent en jeu des disjonctions, il est également important de prendre en compte le nombre de disjonctions du motif.

Pour donner une évaluation de la complexité suivant ces paramètres, on propose dans un premier temps d'étudier la complexité de l'algorithme `getReachable` et la réduction de motifs par $\mathfrak{R}$ (Figure 4.8) en fonction de la taille des termes et des motifs en annotation. L'algorithme `getReachable` présenté en Figure 4.7 étant plus facilement quantifiable, on propose donc de donner sa complexité par rapport à la taille des motifs en annotation comme sur-approximation de celle de l'algorithme implémenté.

- [`getReachable`] La complexité de `getReachable` est directement liée au nombre de nœuds quasi-variables du graphe total considéré. La construction du graphe total a une complexité linéaire avec le nombre de nœuds, l'étude d'instanciabilité a une complexité quadratique et l'analyse d'atteignabilité a une complexité linéaire. De plus le nombre de nœuds quasi-variables est limité par la taille du motif annotant considéré : le motif complément du nœud quasi-variable est une somme de sous-motifs du motif annotant. On a donc une complexité pire cas exponentielle avec le nombre de disjonctions de l'annotation (dont l'effet est limité par le fait que le motif complément doit être bien typé), et en moyenne une complexité polynomiale avec le nombre de symboles constructeurs des annotations (dont l'exposant dépend de l'arité des symboles).
- [$(t \setminus p) \downarrow_{\mathfrak{R}}$] La réduction d'un tel motif se fait principalement via les règles $M7$ et $M8$. Dans le pire cas $M7$ s'applique autant de fois qu'il y a de symboles constructeurs dans le terme t et dans le motif p, en générant un nombre de termes proportionnel au nombre de symboles dans t. De plus, la règle $M6'$ a un effet exponentiel avec le nombre de disjonctions du motif p. Enfin, une fois le motif réduit en forme additif les disjonctions sont remontés par la règle $S1$, qui est appliqué autant de fois qu'il y a de disjonctions. La complexité pire cas de la réduction est donc polynomiale avec le nombre de symboles de t et p, et exponentielle avec le nombre de disjonctions de p. On notera cependant que l'effet polynomial de la règle $M7$ et l'effet exponentiel de la règle $M6'$ sont limités par la taille de la signature, qui tend à favoriser l'application de la règle $M8$. En moyenne, on observe donc une complexité linéaire avec le nombre de symboles de t.
- [$(t \times x_s^{-p}) \downarrow_{\mathfrak{R}}$] La réduction d'un tel motif se fait principalement via la règle $P2$, qui s'applique autant de fois qu'il y a de symboles constructeurs dans le terme t. L'application de cette règle génère également des compléments qui doivent être réduits comme vu dans le cas précédent. Enfin, on notera que pour l'application des règles $P3 - P7$, il faut vérifier la condition en utilisant l'algorithme `getReachable`. La complexité de la réduction est donc polynomiale avec le nombre de symboles de t et p et exponentielle avec le nombre de disjonctions de p.
- [$(t \times p) \downarrow_{\mathfrak{R}}$] La réduction d'un tel motif se fait principalement via les règles $T3$ et $T4$. Dans le pire cas $T3$ s'applique autant de fois qu'il y a de symboles constructeurs dans le terme t et dans le motif p. La règle $P1$ peut également s'appliquer avec une complexité similaire au cas précédent. De plus, la règle $S3$ a un effet linéaire avec le nombre de disjonctions de p. On a donc une complexité linéaire avec le nombre de symboles de t, polynomiale avec le nombre de symboles des annotations de t et linéaire avec le nombre de disjonctions de p.

On peut donc conclure quant à la complexité globale de la méthode d'analyse :

- [**taille des termes du CBTRS**] Pour l'étape d'inférence, on a besoin de faire une réduction de conjonctions de la forme $ls_i \times x_s^{-p}$, avec ls_i un argument du membre gauche de la règle. On a donc une complexité polynomiale avec le nombre de symboles des membres gauches des règles (avec un facteur dépendant du nombre de disjonctions des motifs en annotation). Pour l'étape de vérification de propriétés d'Exemption de Motif, on génère une décomposition de taille linéaire avec le nombre de symboles du terme, et dont le nombre de symboles

total est proportionnel au nombre de symboles du terme initial (avec un facteur dépendant de l'arité des symboles). Pour chaque terme de la décomposition obtenue, on doit ensuite réduire une conjonction de la forme $t \times p$. On a donc une complexité pire cas polynomiale avec le nombre de symboles des membres droits des règles, et une complexité moyenne linéaire.

- [**taille des motifs en annotation**] La taille des motifs en annotation impacte de façon similaire les trois étapes de l'analyse. En effet, l'utilisation de l'algorithme `getReachable` et du système $\mathfrak{R}$ pour réduire des conjonctions conduit à une complexité polynomiale avec le nombre de symboles des motifs et exponentielle avec le nombre de disjonctions.
- [**signature algébrique**] Beaucoup de paramètres de la signature algébrique peuvent entrer en considération, on s'intéresse notamment en au nombre de sortes, au nombre de symboles constructeurs et à l'arité des symboles :
  - [**nombre de sortes**] le nombre de sortes influe principalement sur la taille du graphe total considéré par l'algorithme `getReachable`. Cela dit ce dernier est principalement défini par le motif annotant considéré. Toute chose égale par ailleurs, le nombre de sortes a donc un effet négligeable sur la complexité de la méthode d'analyse.
  - [**nombre de symboles constructeurs**] de même, le nombre de symboles constructeurs influe également sur la taille du graphe total construit et exploré par l'algorithme `getReachable`. Contrairement au nombre de sortes, il impacte cependant la complexité de façon indépendante au motif annotant considéré puisque pour chaque nœud variable exploré, il faut générer des nœuds constructeurs (ou des arrêtes multiples) pour chaque constructeur de la sorte de la variable. L'effet est borné par $O(n^s)$ où s est la taille du graphe total et n le nombre moyen de constructeurs par sorte. Comme les nœuds ne sont explorés qu'une seule fois, en pratique on peut ne considérer que les symboles constructeurs affectés par l'annotation, on a donc une complexité meilleur cas linéaire avec le nombre de symboles constructeurs, et une complexité moyenne polynomiale.
  - [**arité des symboles**] l'arité des symboles a un effet non négligeable sur la complexité de la méthode d'analyse mais difficilement mesurable puisqu'elle impact énormément les paramètres de taille des motifs en annotation et des termes du CBTRS. Toute chose étant égale par ailleurs, l'effet de l'arité des symboles est polynomiale sur la complexité de la réduction suivant $\mathfrak{R}$ (notament au regard des règles M7, T3, P1 et P2) avec un facteur dépendant de la taille des motifs en annotations. Vis-à-vis de l'algorithme `getReachable`, l'augmentation de l'arité des symboles augmente le nombre de nœuds visités avec un effet polynomiale. Un nœud n'étant cependant exploré complètement qu'une seule fois, en moyenne cet effet est négligeable. La méthode d'analyse a donc une complexité polynomiale avec l'arité des symboles, avec une exponentiation assez importante.

Enfin, la méthode d'analyse ne dépend d'aucune hypothèse quant à la terminaison ou la confluence de la relation de réécriture considérée. L'approche s'applique donc à l'analyse de CBTRS n'étant pas nécessairement (fortement) normalisant ou complet, et permet tout de même de garantir la préservation des propriétés d'Exemption de Motif à chaque pas de la réduction induite par le système. En pratique, la méthode prend particulièrement sens quand appliquée à des systèmes au moins faiblement normalisants, et permet alors de garantir des propriétés d'Exemption de Motif vérifiées par les formes normales potentielles de la relation de réécriture.

6.1.2 Cas Non-linéaires

Pour l'analyse statique de CBTRS non-linéaires, on se repose sur la notion de satisfaction de profil par substitution (Définition 5.9) pour vérifier que la relation de réécriture induite préserve la sémantique substitutive (Définition 5.7). Pour cela, la Proposition 5.19 établit qu'il est nécessaire et suffisant de montrer que chaque règle satisfait tous les profils du symbole de tête du membre gauche.

Pour chaque règle et chaque profil de l'annotation du symbole défini en tête de son membre gauche, la méthode d'analyse se présente en deux étapes :

1. une étape d'inférence des substitutions valeurs satisfaisant la pré-condition du profil considérée, basée sur la Proposition 5.21 ;
2. une étape de vérification de la post-condition utilisant la procédure de décision présentée dans la Figure 5.1 pour inférer des contraintes vérifiée par l'algorithme `checkInstance`.

Pour chaque règle, on construit ainsi une forme inférée de la règle satisfaisant la pré-condition donnée par le membre gauche de chaque profil. La méthode d'analyse teste alors, pour chaque règle inférée ainsi obtenue, que la post-condition, donnée par le membre droit du profil, est vérifiée. Pour toute règle inférée ne vérifiant pas sa post-condition, l'implémentation renvoie alors une instance du membre droit construite via la substitution obtenue par la procédure de décision.

Comme pour la méthode d'analyse linéaire, les deux étapes sont dépendantes du système $\mathfrak{R}$ pour réduire des conjonctions de motifs, et dépendent donc de l'implémentation du système et de `getReachable` déjà utilisée pour la méthode linéaire. De façon similaire à l'approche considérée pour l'algorithme `getReachable`, l'algorithme de `checkInstance`, qui permet de vérifier qu'une différence de sémantique de la forme $[\![p]\!] \setminus [\![\sum_{q\in Q} x_s^{-q}]\!]$ est non-vide, synchronise la construction du graphe total et son exploration. Dans l'implémentation, l'approche a simplement été linéarisé pour permettre de renvoyer un terme servant de contre-exemple au fait que la différence soit vide.

Pour la procédure de décision non-linéaire permettant de vérifier la post-condition, plutôt que de générer l'ensemble des toutes les dérivations possibles, l'implémentation construit simplement une table d'association entre les variables/sous-termes du membre droit considéré et les contraintes induites.

Comme pour pour la méthode d'analyse linéaire, on peut donner une caractérisation précise de la complexité de l'analyse linéaire suivant le nombre de règles et le nombre de profils :

- **[nombre de règles du CBTRS]** Comme pour l'analyse linéaire, l'approche non-linéaire vérifie la satisfaction des profils de chaque règle indépendamment du reste du système. La méthod d'analyse opère donc également avec une complexité linéaire par rapport au nombre de règles du système considéré.
- **[nombre de profils des annotations]** Comme pour l'analyse linéaire, il faut vérifier que chaque règle satisfait tous les profils de l'annotation du symbole de tête à gauche de la règle. Du point de vue de la procédure de décision, pour chaque symbole définit, le nombre de dérivations évolue exponentiellement avec le nombre de profils par annotations, puisque les règles ($\mathbf{F}_\setminus$) et ($\mathbf{F}_\times$) considèrent l'ensemble des combinaisons des profils.

La complexité de la méthode d'analyse non-linéaire dépend également de la taille des termes, des motifs et de la signature considérée. On donne une estimation de cette complexité :

- **[taille des termes du CBTRS]** La complexité de l'étape d'inférence pour l'analyse non-linéaire est inchangée : elle est polynomiale avec le nombre de symboles des membres

gauches des règles (avec un facteur dépendant du nombre de disjonctions des motifs en annotation).

Pour l'étape de vérification de propriétés d'Exemption de Motif, la procédure de décision (Figure 5.1) n'admet qu'un nombre limité de dérivations pour chaque symbole du membre droit. On a donc un nombre de dérivations proportionnel au nombre de symboles des termes considérés, et l'ensemble des dérivations (*i.e.* des tables de contraintes, du point de vue de l'implémentation) peut ainsi être généré avec une complexité linéaire au nombre de symboles. La complexité de la méthode d'analyse est donc polynomiale avec le nombre de symboles des termes du CBTRS, dans le pire cas, et linéaire en moyenne.

- [**taille des motifs en annotation**] La complexité de l'étape d'inférence pour l'analyse non-linéaire est inchangée : elle est polynomiale avec le nombre de symboles des motifs et exponentielle avec le nombre de disjonction. Pour l'étape de vérification des post-conditions, on peut observer qu'avec la règle ($\mathbf{C}^{+}_{\times}$), le nombre et la taille des dérivations de la procédure de décision (Figure 5.1) augmente également linéairement avec le nombre de disjonctions des motifs. De plus, les contraintes d'instanciations données par le contexte obtenue par la procédure de décision dépendent également des motifs en annotation. La vérification de ces contraintes par l'algorithme `checkInstance` se reposant sur la notion de graphe total utilisée par `getReachable`, on a donc une complexité similaire : exponentielle avec le nombre de disjonctions, dans le pire cas, et polynomiale avec le nombre des symboles des motifs.

- [**signature algébrique**] Du point de vue de la signature algébrique, la complexité de la méthode d'analyse non-linéaire est comparable à celle de l'approche linéaire. En effet, l'étape d'inférence est identique entre les deux approches et le calcul des dérivations de la décision de procédure peut se ramener aux calculs de sémantique considérés dans l'analyse linéaire. La différence se fait principalement dans la gestion des profils en annotation des symboles définis. On a donc une complexité moyenne polynomiale avec le nombre et l'arité des symboles constructeurs. Le nombre de sortes a un effet négligeable.

Enfin, on rappelle que bien que la méthode d'analyse soit applicable indépendamment des propriétés de terminaison et de confluence du CBTRS considéré, la propriété de préservation de sémantique n'est particulièrement intéressante qu'avec certaines de ces hypothèses. En effet, contrairement à l'approche linéaire, l'analyse non-linéaire ne permet pas de démontrer la préservation de la sémantique à chaque pas de réduction mais garantit que la préservation est éventuellement obtenue sur plusieurs pas (cf. Proposition 5.16). Comme montré dans l'Exemple 5.14, la préservation de la sémantique substitutive garantit donc l'existence d'une dérivation préservant la sémantique du terme initial, mais l'absence de confluence et/ou de terminaison (au moins faible) peut ainsi se traduire en l'existence de dérivations ne la préservant pas.

Ces dépendances à des propriétés de confluence et de terminaison de la relation de réécriture considérée, ainsi que la complexité plus importante de la méthode d'analyse rendent en général l'approche non-linéaire plus difficilement applicable que l'analyse linéaire.

6.2 Optimisations

On présente certains choix d'implémentation destinés à optimiser les performances de la méthode d'analyse.

6.2.1 Mise en cache

La méthode d'analyse proposée, en particulier pour l'approche linéaire, a tendance à être très redondante. En effet, que ce soit pour la construction de l'équivalent sémantique ou la vérification de la post-condition, on calcule des intersections de sémantique de la forme $\{\!|\tau|\!\} \cap [\![p]\!]$, avec τ une version aliassée d'un terme et p un motif additif linéaire et régulier.

Exemple 6.2. *Dans l'algèbre de termes définie par la signature Σ_{list} introduite dans l'Exemple 3.5, on a présenté en Section 4.5 l'application de la méthode pour l'analyse du système $\mathcal{R}$ suivant :*

$$\begin{cases} \mathit{flatten}(\mathit{nil}) & \to & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \to & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \to & \mathit{concat}(\mathit{flatten}(l), \mathit{flatten}(l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \to & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \to & l \end{cases}$$

où les symboles définis $\mathcal{D} = \{\mathit{flatten}^{!\mathcal{P}_1} : \mathsf{List} \mapsto \mathsf{List}, \mathit{concat}^{!\mathcal{P}_2} : \mathsf{List} * \mathsf{List} \mapsto \mathsf{List}\}$ sont annotés avec $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ et $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$ où $p_{flat} = \mathit{cons}(\mathit{lst}(l_1), l_2)$.

En considérant les calculs d'équivalents sémantiques et les vérifications de post-conditions, l'analyse a nécessité le calcul des intersections suivantes (en ignorant les cas où le motif est $\bot$) :

- $\{\!|\mathit{nil}|\!\} \cap [\![p_{flat}]\!]$ *pour la première règle ;*
- $\{\!|\mathit{cons}(\mathit{cons}(\mathit{int}(n \mathbin{@} y^{-\bot}_{\mathsf{Int}}), l \mathbin{@} z^{-p_{flat}}_{\mathsf{List}}))|\!\} \cap [\![p_{flat}]\!]$ *pour la deuxième règle ;*
- $\{\!|l \mathbin{@} z^{-p_{flat}}_{\mathsf{List}}|\!\} \cap [\![p_{flat}]\!]$ *deux fois pour le calcul de l'équivalent sémantique de la troisième règle, une fois pour la vérification de la post-condition de cette même règle, deux fois pour le calcul de l'équivalent sémantique de la quatrième règle, et pour la vérification de la post-condition de la dernière règle.*
- $\{\!|\mathit{cons}(e \mathbin{@} (x^{-p_{flat}}_{\mathsf{Expr}} \setminus \mathit{lst}(l_1)), l \mathbin{@} z^{-p_{flat}}_{\mathsf{List}})|\!\} \cap [\![p_{flat}]\!]$ *pour la quatrième règle.*

On voit clairement qu'il y a beaucoup de calculs redondants de l'intersection $\{\!|l \mathbin{@} z^{-p_{flat}}_{\mathsf{List}}|\!\} [\![p_{flat}]\!]$ (qui est également calculée pour deux des trois autres intersections considérées, pour un total de huit fois).

Bien que certaines de ces redondances puissent être éliminées localement, dans le cas des calculs d'équivalents sémantiques, la méthode peut clairement bénéficier d'une approche permettant de limiter les calculs redondants. Dans ce but, l'implémentation utilise un cache permettant de stocker les résultats de ces calculs d'intersection.

Ainsi, pour un faible surcout de mémoire, cette approche permet une amélioration de performance en termes de temps de calcul assez significative. On peut en effet observer avec les performances données dans la Figure 6.2, que la vérification d'une seule règle de réécriture générée aléatoirement avec un terme droit contenant 100 symboles constructeurs et un motif d'annotation contenant 50 symboles constructeurs (d'arité 6) prend, en moyenne, $40ms$ ($47ms$ avec une analyse stricte). L'analyse d'un CBTRS de 25 règles ne nécessite, grâce à l'approche de *caching*, que $486ms$ en moyenne ($542ms$ avec une analyse stricte), soit un gain de 51% par rapport aux temps estimé par rapport à l'analyse d'une seule règle.

Pour l'approche non-linéaire, l'analyse ne calcule plus directement d'intersection de sémantiques, mais se repose sur la procédure de décision présentée de Figure 5.1 mais infère des contraintes sur l'évaluation des membres droits. Une approche similaire de *caching* est donc utilisée pour optimiser les performances de la méthode d'analyse en stockant les résultats de satisfaisabilité de ces contraintes. Les bénéfices de l'approche de *caching* pour l'analyse non-linéaire

sont cependant moins claires puisque la différence de performance entre un CBTRS de 25 règles et une seule règle est négligeable (donc suffisante pour compenser le coût d'écriture et de lecture du cache, mais pas suffisamment pour justifier le surcout mémoire). Cela dit l'effet, on observe un gain de temps performance significative sur les scénarios de tests fournis : le cache permet une réduction du temps de calcul de 36% en moyenne sur l'ensemble des scénarios fournis.

6.2.2 Optimisation de linéarité

Entre la complexité bien plus importante de l'approche non-linéaire et l'efficacité moindre de l'optimisation par *caching*, une analyse complètement non-linéaire n'est généralement pas rentable du point de vue de ses performances.

Comme présenté dans le Chapitre précédent, l'implémentation propose différentes options pour l'analyse d'un système de réécriture :

- l'approche linéaire classique, qui dans le cas d'un système non-linéaire fonctionne par linéarisation pour sur-approximer la sémantique des termes non-linéaires ;
- l'approche stricte qui analyse le CBTRS sous une hypothèse de stratégie de réduction stricte pour utiliser les propriétés de préservation par substitution valeur ;
- l'approche non-linéaire complète, qui analyse l'ensemble des règles du CBTRS via la méthode non-linéaire présentée dans le Chapitre précédent ;
- l'approche agile par défaut, qui tente de déterminer, pour chaque règle si l'utilisation de la méthode non-linéaire est nécessaire pour vérifier la satisfaction des profils.

Un tableau décrivant les performances de chacune des options d'analyse est présenté en Figure 6.3.

Pour l'approche agile, elle propose de déterminer si la sémantique substitutive du membre droit d'une règle est différente de sa sémantique close. Si c'est le cas, l'utilisation de cette dernière peut donc permettre de vérifier la satisfaction d'un profil qui pourrait être rejetée par l'approche classique. La règle est alors analysée par la méthode non-linéaire. Sinon, on utilise une approche stricte confluente, basée sur la sémantique confluente, comme présentée dans [CLM21]. Dans le cas linéaire, cette dernière est identique à la méthode présentée dans le Chapitre 4, et dans le cas non-linéaire, elle est équivalente à l'approche stricte avec une hypothèse de confluence supplémentaire.

Enfin, étant donné que les mécanismes de *caching* linéaire et non-linéaire sont différents, comme l'approche agile privilégie généralement une analyse linéaire, elle utilise donc préférentiellement un cache linéaire. Pour les règles analysées avec l'approche non-linéaire, un cache local à l'analyse de la règle est alors utilisé. Cela explique certaines différences de performance entre l'approche agile et l'approche non-linéaire : la première étant censée choisir l'approche optimale, elle est en général plus performante que l'approche non-linéaire (ou dans une marge négligeable), l'utilisation non-optimale du cache non-linéaire peut cependant impacter significativement les performances quand beaucoup de règles sont analysées avec l'approche non-linéaire.

6.3 Comparaison à l'état de l'art

On propose finalement de comparer les résultats obtenus par notre approche d'analyse, ainsi que ses performances, avec d'autres approches présentées dans la Section 2.3.

	analyse agile	analyse linéaire	analyse stricte	analyse non-linéaire
flatten1	$25\mu s$	$23\mu s$	$23\mu s$	$42\mu s$
flatten2	$46\mu s$	$41\mu s$	$42\mu s$	$99\mu s$
flatten3	$42\mu s$	$37\mu s$	$35\mu s$	$64\mu s$
flatten fail	$35\mu s$	$31\mu s$	$28\mu s$	$56\mu s$
negativeNF	$392\mu s$	$387\mu s$	$359\mu s$	$637\mu s$
skolem	$82\mu s$	$47\mu s$	$47\mu s$	$121\mu s$
removePlus0 fail	$29\mu s$	$26\mu s$	$23\mu s$	$61\mu s$
removePlus0	$53\mu s$	$50\mu s$	$50\mu s$	$98\mu s$
multiply0	$28\mu s$	$11\mu s$	$12\mu s$	$23\mu s$
insertionSort	$478\mu s$	$201\mu s$*	$195\mu s$*	$405\mu s$
mergeSort	$9.26ms$	$847\mu s$*	$569\mu s$*	$8.62ms$
reverse	$134\mu s$	$125\mu s$	$128\mu s$	$171\mu s$
reverse2	$382\mu s$	$360\mu s$	$370\mu s$	$537\mu s$
delete	$446\mu s$	$113\mu s$*	$108\mu s$*	$415\mu s$
sortedDelete	$884\mu s$	$265\mu s$*	$228\mu s$*	$738\mu s$
otrs	$3.67ms$	$1.4ms$	$1.4ms$	$3.08ms$
TRS aléatoire	$490ms$	$486ms$	$542ms$	$1,36s$
terme aléatoire	$47.5ms$	$40ms$	$47ms$	$54.7ms$

FIGURE 6.3 – Différences de performance en temps d'exécution entre les différentes approches d'analyse ; les scénarios *TRS aléatoire* et *terme aléatoire* correspondent à la vérification d'un TRS de 25 règles et d'une propriété d'Exemption de Motif d'un terme ; les autres scénarios sont présentés en Appendix C, avec ceux marqués * modifiés pour pouvoir être validés avec les analyses linéaires et strictes.

Complétion d'automate

L'approche dont les résultats peuvent le plus facilement être comparés à ceux obtenus par notre méthode d'analyse est celle de complétion d'automates [GR10], qui a donné lieu à l'outil Timbuk [Hau20]. L'utilisation d'automates d'arbres permet notamment de fournir une approche plus expressive que celle présentée ici. En effet, notre approche se concentre principalement sur l'étude de transformation de programmes, comme proposé dans la problématique initiale, et se limite donc à la vérification de garanties syntaxiques pouvant être exprimées sous la forme de propriété d'Exemption de Motif. L'automate obtenu par la technique de complétion fournit, lui, une description plus précise des termes et de leurs formes normales, notamment dans le contexte de l'étude de programmes fonctionnels.

Il n'est ainsi pas possible d'exprimer de propriétés de parité des entiers naturels par Exemption de Motif, contrairement à l'approche par automate d'arbre, comme vu dans l'Exemple 2.3. De même, on peut décrire des arbres ordonnés par un automate, mais pas par Exemption de Motif :

Exemple 6.3. *On considère des arbres binaires de valeurs A et B avec $A \leq B$. On peut représenter cette famille d'arbres via l'algèbre de termes définies par la signature $\Sigma = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébrique :*

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{Val} & := & A & \mathsf{Tree} & := & \mathit{leaf} \\ & | & B & & | & \mathit{node}(\mathsf{Val}, \mathsf{Tree}, \mathsf{Tree}) \end{array}$$

L'approche par automate d'arbre permet de caractériser les arbres ordonnés via l'automate $\mathcal{A}$

défini par le système de transition suivant :

$$\begin{cases} A & \rightarrow & q_a \\ B & \rightarrow & q_b \\ node(q_a, q_{ta}, q_{ta}) & \rightarrow & q_{ta} \\ node(q_b, q_{tb}, q_{tb}) & \rightarrow & q_{tb} \\ node(q_a, q_{ta}, q_{tab}) & \rightarrow & q_{tab} \\ node(q_b, q_{tab}, q_{tb}) & \rightarrow & q_{tab} \\ leaf & \rightarrow & q_{ta} \\ leaf & \rightarrow & q_{tb} \\ leaf & \rightarrow & q_{tab} \end{cases}$$

Dans cet automate, l'état q_{ta}, respectivement q_{tb}, représente l'ensemble des arbres ne contenant que des valeurs A, respectivement B. Et l'état q_{tab} représente donc l'ensemble des arbres ordonnés.

La technique d'automate d'arbre permet donc de vérifier un certain nombre de garanties sur des transformations d'arbres ordonnés (insertion dans un arbre ordonné, transformation en liste triée...).

En revanche, il n'est pas possible de décrire l'ensemble des arbres ordonnés par une propriété d'Exemption de Motif. Par exemple, l'exemption du motif $node(A, node(B), x)$ garantit que la branche gauche d'un arbre étiqueté A, ne sera pas étiqueté B mais ne permet pas de garantir qu'une branche droite de cette branche ne contiendra pas une valeur B. Cette propriété d'exemption de motif n'est donc pas suffisante pour décrire les arbres ordonnés.

L'approche d'analyse par Exemption de Motif ne permettant de caractériser ces arbres ordonnés, elle ne peut pas vérifier des garanties sur leurs transformations.

La technique d'analyse par complétion d'automates est donc théoriquement plus expressive que notre approche. Il est également important de noter que la complétion d'automates peut également s'appliquer à des programmes fonctionnels d'ordre supérieur. On conjecture cependant qu'un formalisme similaire devrait permettre d'appliquer l'approche par Exemption de Motif à l'étude de fonctions d'ordre supérieur.

Néanmoins, une limitation majeure de la technique de complétion d'automates est que sa terminaison n'est pas garantie. Certaines classes de TRS et de systèmes d'équations pour lesquels cette terminaison est garantie ont été identifiées [Gen16], mais peuvent donc limiter les applications pratiques de l'approche. Contrairement à l'approche par complétion d'automates qui est ainsi dépendante certaines hypothèses sur la stratégie d'évaluation en *Appel par valeur*, et sur la terminaison et la complétude du système, notre approche n'est, dans le cas d'une analyse linéaire ou par linéarisation, dépendante d'aucune hypothèse.

On fournit en Figure 6.4, un tableau de comparaison des performances, en termes de temps d'exécution, de notre implémentation avec Timbuk 3.2 [GBB+17] et Timbuk 4 [Hau20] [1]. On notera qu'on ne présente ici que des scénarios pour lesquels les propriétés considérées peuvent être exprimées par des propriétés d'Exemption de Motif, pour un horizon plus complet de scénarios pouvant être étudiés via l'approche par complétion d'automates, nous renvoyons le lecteur vers les scénarios fournis avec les outils utilisés.

Il est également important de remarquer que, contrairement à notre analyse, l'approche par complétion d'automates n'est que faiblement dépendante d'interventions de l'utilisateur : en effet, dans le cadre de la complétion d'automates, il est possible de fournir un ensemble d'équations permettant de définir des classes d'abstraction, mais l'analyse peut également être effectuée

1. on a fait le choix d'inclure Timbuk 3.2, puisque, bien qu'il s'agisse d'une version plus ancienne, il obtient généralement de meilleurs performances en termes de temps d'exécution

	pfree check		timbuk 3.2		timbuk 4	
flatten1	✓	$25\mu s$	✗	∞	✓	$685ms$
flatten2	✓	$46\mu s$	✗	∞	✓	$975ms$
flatten3	✓	$42\mu s$	✓	$2,3ms$	✓	$1,3s$
negativeNF	✓	$392\mu s$	✓	$3,2ms$	✓	$104s$
skolem	✓	$82\mu s$	✗	$1,5s$	✓	$1,6s$
removePlus0	✓	$53\mu s$	✗	$55ms$	✗	∞
multiply0	✓	$28\mu s$	✓	$2,4ms$	✓	$225ms$
insertionSort	✓	$478\mu s$	✓	$65ms$	✓	$731ms$
mergeSort	✓	$9.26ms$	✗	∞	✓	$1,4s$
reverse	✓	$134\mu s$	✓	$614ms$	✓	$1,4s$
reverse2	✓	$382\mu s$	✓	$714ms$	✓	$2,3s$
delete	✓	$446\mu s$	✓	$2,2ms$	✓	$286ms$
sortedDelete	✓	$884\mu s$	✗	∞	✓	$798ms$
otrs	✓	$3.67ms$	✗	$44ms$	✗	∞

FIGURE 6.4 – Table de comparaison avec Timbuk 3.2 and Timbuk 4. ✓ indique que le scénario est correctement vérifié, ✗ indique un échec de la validation du scénario

indépendamment de toute intervention (en particulier, dans le cas de Timbuk 4, qui utilise une procédure CEGAR pour générer automatiquement ces abstractions). En revanche, la définition de profils en annotation des symboles de fonction est absolument nécessaire pour l'analyse par Exemption de Motif. On stipulera cependant que l'utilisation de propriétés de filtrage par motif facilite grandement l'écriture et la compréhension de ces profils.

Exemple 6.4. *On considère l'algèbre de termes définie par la signature Σ_{list} introduite dans l'Exemple 3.5.*

Contrairement à l'approche par Exemption de Motif qui permet de caractériser les listes et les expressions plates (i.e. entiers ou listes plates) simplement via le motif $cons(lst(l_1), l_2)$, pour l'approche par automate, il est nécessaire de définir l'automate décrit par le système de transition suivant :

$$\begin{cases} z & \to & q_n \\ s(q_n) & \to & q_n \\ int(q_n) & \to & q_i \\ nil & \to & q_l \\ cons(q_i, q_l) & \to & q_l \\ lst(q_l) & \to & q_e \\ q_i & \to & q_e \end{cases}$$

Dans cet automate, l'état q_n décrit les entiers de Peano, l'état q_i décrit le sous-ensemble d'expressions entières, q_l décrit les listes plates, et l'état q_e décrit les expressions plates.

Enfin, on notera que l'approche par complétion d'automates, afin de pouvoir construire une abstraction finie des termes considérés, nécessite généralement que la sorte finale des termes étudiés soient un ensemble de constantes. Il n'est donc pas possible de vérifier directement la forme des termes considérées en donnant l'automate d'arbre attendu : la propriété recherchée doit donc généralement être exprimée via une fonction de vérification, encodée dans le TRS fourni.

Higher-order recursion scheme

La comparaison avec des solutions basées sur la notion de *Higher-order recursion scheme* (HORS)

est, elle, plus compliquée. En effet, même si le principe de l'analyse repose toujours sur la construction d'une abstraction des résultats de la transformation étudiée, l'expressivité des différentes approches basées sur la notion d'HORS est très différente de celle de notre méthode d'analyse. En particulier, on peut mentionner le *model-checker* MoCHi [KSU11, SKU+20] qui est particulièrement adapté à l'étude de programmes manipulant des types de données d'entier et de liste, mais qui reste limité par le formalisme d'HORS pour des types d'arbres algébriques plus généraux.

En termes d'expressivité, on peut tout de même évoquer la solution présentée dans [UTK10], qui propose une approche similaire basée sur l'annotation de *Higher-order multi-parameter tree transducers* (HMTT). Ces annotations et les prédicats considérés sont définis par des automates d'arbres, d'où une expressivité théorique similaire à celle de Timbuk. Malheureusement, aucune implémentation de la méthode n'est publiquement disponible, donc il n'est pas possible de comparer directement les performances, mais au vu des résultats publiés dans [UTK10], notre approche semble tout de même plus rapide. Comme pour Timbuk, cette approche a l'avantage de pouvoir s'appliquer sur des programmes d'ordre supérieur, mais l'utilisation du formalisme de réécriture et l'annotation directe des symboles définis pour l'étude de CBTRS semble généralement être une approche plus communément utilisée et plus facile à appréhender que l'utilisation d'HMTT étendu proposée par [UTK10].

Enfin, on peut s'intéresser à des approches à base d'HORS proposant une déduction automatique des abstractions comme les *Pattern-Matching Recursion Scheme* (PMRS) [OR11]. Dans ce formalisme, la méthode de vérification propose d'utiliser une procédure CEGAR pour affiner l'abstraction. Cependant l'approche utilisée se base sur l'utilisation de motifs finis qui ne permettent souvent pas d'obtenir une abstraction adaptée à la vérification de propriétés comme celles d'Exemption de Motif. De plus l'utilisation du formalisme de PMRS reste généralement plus fastidieuse que celle de CBTRS.

Exemple 6.5. *On considère l'algèbre de termes définie par la signature Σ_{list} introduite dans l'Exemple 3.5. On propose d'encoder le CBTRS suivant, via un PMRS :*

$$\begin{cases} \mathit{flatten}(\mathit{nil}) & \rightarrowtriangle & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \rightarrowtriangle & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \rightarrowtriangle & \mathit{concat}(\mathit{flatten}(l), \mathit{flatten}(l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \rightarrowtriangle & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \rightarrowtriangle & l \end{cases}$$

En passant les différents symboles à l'ordre supérieur, on obtient pour cela, avec les primitives de filtrage limitées du formalisme, un PMRS de la forme :

$$\begin{array}{lcl} \mathit{Flatten}\ \mathit{nil} & \Longrightarrow & \mathit{nil} \\ \mathit{Flatten}\ (\mathit{cons}\ e\ l) & \Longrightarrow & \mathit{Check}\ l\ e \\ \mathit{Check}\ l\ (\mathit{int}\ i) & \Longrightarrow & \mathit{cons}\ (\mathit{int}\ i)\ l \\ \mathit{Check}\ l\ (\mathit{lst}\ l') & \Longrightarrow & \mathit{Concat}\ (\mathit{Flatten}\ l')\ (\mathit{Flatten}\ l)) \\ \mathit{Concat}\ l'\ (\mathit{cons}\ e\ l) & \Longrightarrow & \mathit{cons}\ e\ (\mathit{Concat}\ l'\ l) \\ \mathit{Concat}\ l'\ \mathit{nil} & \Longrightarrow & l' \end{array}$$

De plus, le résultat de l'analyse proposée par ces approches est une caractérisation des formes normales obtenues par transformation, d'où une dépendance à des hypothèses de terminaison, et généralement de confluence, de la réduction considérée. Dans notre cas, la terminaison faible est nécessaire pour pouvoir en pratique décrire les formes normales considérées, mais la méthode d'analyse permet de garantir la préservation de sémantique du système de réécriture étudié indépendamment de sa terminaison.

Une dernière contribution [MKU15], apparentée au formalisme d'HORS, propose une approche totalement automatisée pour la vérification de programmes fonctionnels. Cette dernière repose sur une approche assez gloutonne qui consiste à diviser systématiquement l'ensemble des états de l'automate d'entrée en un nombre croissant de sous-états, tant qu'un résultat faux-négatif est identifié. Cette méthode a l'avantage de proposer un résultat de complétude relative. Aucune implémentation n'étant publiquement disponible, on remarque donc simplement que cette approche gloutonne semble cependant moins performante que l'approche par complétion d'automates proposée par Timbuk 4 [HGJ20].

Approches de vérification par typage
Dans le Chapitre 2, on a présenté des solutions de typage se reposant sur l'utilisation d'annotations pour décrire des spécifications des fonctions à vérifier. Cependant, les approches citées ont généralement une expressivité difficilement comparable à celle de notre analyse par Exemption de Motif. En particulier, les approches basées sur la notion de *Refinement Types* [FP91], comme l'utilisation de *Liquid Types* en Haskell, sont plus adaptées à l'étude de propriétés sur des types de données intégrés à la sémantique d'annotation (comme des entiers ou des listes). Néanmoins, ces approches permettent généralement d'exprimer des propriétés algébriques locales, mais ne sont pas adaptés pour l'étude de propriétés plus structurelles des types algébriques, tel que l'Exemption de Motif.

Une autre approche se reposant sur des notions de typage complexes pour garantir certaines propriétés de correction des transformations est celle proposée par CDuce [BCF03]. Ce langage fonctionnel destiné à l'écriture de transformation XML propose en effet un système de typage fort associant des notions de polymorphisme, d'inférence de type et de compositions de types avec une approche de sous-typage par sémantique assez proche de l'approche de satisfaction de sémantique utilisée par notre méthode d'analyse. Cette particularité du langage permet une vérification statique des spécifications ainsi exprimées. Bien que cette approche soit très expressive, la méthode de sous-typage par sémantique semble assez superficielle puisqu'elle échoue, néanmoins, à vérifier correctement la majorité de nos scénarios (présentés en Annexe C). Notamment, contrairement à notre méthode, l'approche proposée ne permet généralement pas de prendre en compte les contraintes de corrélation dans les cas non-linéaires. Enfin, bien que le langage se présente comme généraliste, il reste particulièrement ancré dans le contexte des transformations XML, qui, bien qu'ayant de très nombreuses applications dans les domaines industriels, ne s'accommode pas spécialement à l'étude des langages fonctionnels.

6.4 Synthèse

On a proposé une implémentation publiquement disponible et testable en ligne de la méthode d'analyse présentée. L'implémentation est accompagnée d'un jeu de fichiers de tests, dont les scénarios sont décrits en Annexe C. À l'exception de l'algorithme `getReachable`, l'implémentation est assez fidèle aux différents mécanismes introduits dans les Chapitres précédents. En se basant sur la même approche théorique, la version implémentée de `getReachable` (Figure 6.1), obtient non seulement de meilleures performances, mais permet également de prendre en compte des termes potentiellement infinis, qui peuvent être intéressants pour l'étude de programmes dans un langage *lazy*. Une autre optimisation majeure proposée consiste à utiliser un cache afin de limiter la redondance des calculs effectués. En termes de temps de calcul, l'utilisation de ce cache permet en effet d'améliorer les performances moyennes de l'implémentation d'un facteur 2.

L'implémentation propose à l'utilisateur différentes options d'analyse correspondants aux

approches linéaires et non-linéaires présentées dans les Chapitres précédents. En termes de complexité, on observe une complexité linéaire avec le nombre de règles, polynomiale avec le nombre de symboles des membres gauches et linéaire (en moyenne) avec le nombre de symboles des membres droits. De façon similaire, les motifs en annotation ont un effet polynomial avec le nombre de symboles et un effet exponentiel avec le nombre de disjonctions. La majeure différence, en termes de complexité, entre la méthode linéaire et la méthode non-linéaire se mesure au niveau du nombre de profils par annotation, qui a un impact polynomial pour les cas linéaires et exponentiel pour les cas non-linéaires. Les effets polynomiaux ont aussi généralement des poids plus importants dans le cas non-linéaire.

Enfin, l'implémentation présente des résultats assez encourageant par rapport aux autres approches de la littérature. En particulier, dans le contexte de l'analyse de transformations de programmes, l'utilisation de notre formalisme de système d'annotations est faiblement intrusive et assez intuitive, grâce à l'aspect algébrique des propriétés de filtrage considérées, et donne de très bons résultats en termes de performance. En termes d'expressivité, un certain nombre d'approches alternatives propose des formalismes plus particulièrement adaptés à la validation de programmes manipulant des entiers ou des listes, pour lesquels ils sont plus expressifs que notre formalisme d'Exemption de Motif. Dans le contexte de la programmation fonctionnelle, d'autres approches utilisent des formalismes, comme celui d'automates d'arbres, plus expressifs mais généralement plus difficiles à appréhender.

Conclusion

Dans ce mémoire, on a introduit une approche originale, basée sur le formalisme de réécriture, pour exprimer et vérifier des spécifications algébriques de programmes. L'objectif étant de pouvoir garantir que le résultat d'un programme, décrit par un système de réécriture, ne contient pas de sous-terme filtré par certains motifs spécifiques éliminés, on a proposé un formalisme permettant d'exprimer ce type de garanties, et on a réalisé une étude de ce formalisme dans le contexte d'une réduction par relation de réécriture. Cette étude a notamment conduit à l'élaboration d'une méthode d'analyse statique permettant de vérifier systématiquement les spécifications exprimées par le formalisme. Une implémentation en Haskell de cette méthode a été développée et testée, pour démontrer la pertinence de l'approche proposée.

Formalisme d'Exemption de Motif

Nous avons proposé un formalisme d'annotation des symboles de fonction reposant sur la notion d'Exemption de Motif qui décrit, algébriquement, la propriété que l'on cherche à garantir sur les résultats de la transformation : aucun sous-terme n'est filtré par un certain motif que la fonction est censée éliminer. Cela se traduit donc sous la forme de spécifications algébriques définies par un ensemble de profils annotant les symboles de fonctions. Chaque profil donné en annotation décrit, en termes d'Exemption de Motif, une pré-condition sur la forme des arguments de la fonction qui, quand elle est respectée, doit garantir une post-condition similaire sur le résultat obtenu.

Ces spécifications peuvent donc être utilisées pour donner une sur-approximation des résultats attendus d'une telle application de fonction. Pour formaliser cette sur-approximation, on a présenté une généralisation de la notion de sémantique close introduite dans [CM19]. Cette dernière permet de représenter la potentielle infinité de valeurs filtrées par un motif dans une structure finie permettant le calcul d'opérations basées sur les combinateurs ensemblistes classiques. En se servant des annotations des symboles de fonctions, on obtient ainsi une sur-approximation plus ou moins précise, en fonction des profils spécifiés, qui peut être utilisée pour vérifier qu'un système de réécriture, décrivant le comportement des fonctions considérées, respecte les spécifications associées.

En effet, du point de vue du formalisme de réécriture, on a montré que les spécifications, décrites par le système d'annotations choisi, peuvent être vérifiées à l'échelle d'une règle de réécriture. Chaque règle donne une définition algébrique du comportement des fonctions considérées, qui peut ainsi être comparée à la pré-condition et à la post-condition de chaque profil annotant le symbole des fonctions correspondantes. La notion de sémantique proposée permet donc de vérifier que, dans les cas où la règle respecte la pré-condition d'un profil, la sur-approximation obtenue respecte la post-condition associée : on dit alors que la règle satisfait le profil.

Méthode d'analyse statique

Prouver la correction des spécifications algébriques décrites par le système d'annotations, revient donc à vérifier que toutes les règles du système de réécriture considéré satisfont les profils de l'annotation du symbole en tête du membre de gauche : respecter la pré-condition du profil par instanciation du membre gauche de la règle implique de respecter la post-condition avec la même instanciation du membre droit de la règle. Pour cela, on a proposé une méthode d'analyse statique, qui se présente en trois étapes :

1. une étape d'inférence des substitutions satisfaisant la pré-condition du profil considéré ;
2. une étape de construction de sémantique, qui reconstruit une sur-approximation des instances correspondantes du membre droit de la règle ;
3. une étape de vérification, qui vérifie que cette sémantique exprimant la sur-approximation des résultats respecte la post-condition donnée par le profil.

Les trois étapes reposent sur différents mécanismes permettant l'étude de la sémantique des termes. En utilisant une représentation des sémantiques sous la forme de graphe, on a proposé un algorithme permettant de vérifier si une sémantique est vide et, dans le cas contraire, d'exprimer l'ensemble des sous-termes de cette sémantique. Grâce à cet algorithme, on a pu proposer un système de réduction des motifs étendus qui préserve la sémantique des motifs considérés. Ce système est notamment utilisé afin de comparer des motifs entre eux, tout en exprimant d'éventuelles contraintes d'instanciation résultant de cette comparaison. Ces différents outils permettent non seulement l'étude des sémantiques, mais également de vérifier systématiquement des propriétés d'Exemption de Motif.

L'ensemble de ces mécanismes permet donc de proposer une méthode d'analyse statique pour vérifier, par satisfaction de profil, que le comportement du programme décrit par un système de réécriture respecte les spécifications algébriques données par le système d'annotations. La méthode ne dépend d'aucune hypothèse quant à la terminaison ou la confluence de la relation de réécriture, en garantissant qu'à chaque pas de réécriture la sémantique du terme réduit et, par conséquent, ses propriétés d'Exemption de Motif sont préservées. Dans le cas de systèmes non-linéaires, elle ne s'applique cependant qu'avec certaines limitations.

Analyse de systèmes non-linéaires

On a montré que la méthode d'analyse proposée ne s'applique aux systèmes non-linéaires que par sur-approximation via un système linéaire, ou sous hypothèse d'une relation de réécriture stricte, donc dans le cadre d'un programme avec une stratégie d'évaluation en *Appel par valeur*. Pour éviter ces limitations, on a proposé et étudié différents formalismes permettant de prendre en compte les contraintes de corrélation des différentes instances des variables d'un terme non-linéaire.

Le formalisme retenu propose une notion de sémantique substitutive qui donne une sur-approximation des formes normales potentielles d'un terme en supposant que deux sous-termes identiques seront réduits de manière équivalente. Cette approche revient à faire une hypothèse de confluence de la relation de réécriture, qui dans le cadre de la programmation fonctionnelle est généralement garantie par déterminisme. Pour l'analyse statique de systèmes de réécriture, on a montré que cette nouvelle notion de sémantique induit une notion de satisfaction de profil proche de celle de la sémantique close utilisée pour l'analyse de systèmes linéaires. On peut donc vérifier la correction des spécifications algébriques décrites par le système d'annotations en montrant que chaque règle du système considéré satisfait localement les profils en annotation.

En réutilisant certains des mécanismes déjà présentés, on a ainsi pu proposer une adaptation de la méthode d'analyse précédente. La majeure différence réside dans le fait que, pour l'analyse de termes non-linéaires, l'étape de construction de la sur-approximation sémantique n'est pas assez précise pour prendre en compte les contraintes liées à la non-linéarité. Pour vérifier la satisfaction de la post-condition, on utilise alors une procédure de décision permettant de vérifier l'existence d'une instance non-conforme à la post-condition. La méthode ainsi proposée permet donc, sous hypothèse de confluence et avec un surcoût en termes de performance, l'analyse statique de systèmes non-linéaires quand les approches linéaire ou stricte ne sont pas suffisantes.

Perspectives

Les perspectives de l'approche proposée sont principalement liées à l'amélioration du formalisme et/ou de la méthode d'analyse. La comparaison à l'état de l'art donne des pistes assez claires sur les évolutions possibles : en particulier l'extension du formalisme à des formes d'annotations plus variées pour améliorer l'expressivité, et le support de fonctions d'ordre supérieur. Une autre amélioration envisagée concerne l'automatisation de la méthode d'analyse via un mécanisme d'inférence de profils, afin de limiter le travail d'annotation de l'utilisateur. Enfin, une intégration plus importante aux outils basés sur la réécriture stratégique, et notamment au langage Tom, est également une piste d'évolution importante.

Expressivité

La principale piste d'évolution de notre formalisme concerne son expressivité. Bien que l'approche proposée réponde parfaitement à la problématique initialement considérée, et est donc particulièrement bien adaptée à l'étude de transformations de programmes, dans un contexte plus général, elle peut être limitée par le formalisme d'Exemption de Motif. Comme les annotations des symboles sont définis sous la forme de profils exprimant des spécifications algébriques basées sur des propriétés d'Exemption de Motif, notre approche ne peut ni exprimer, ni vérifier des propriétés qui ne peuvent pas être décrites via l'Exemption de Motif. C'est, par exemple, le cas de la notion d'arbre ordonné, de propriétés arithmétiques des entiers comme la parité, ou encore des contraintes de taille sur des listes.

En pratique, la notion de sémantique de motif utilisée et les différents mécanismes mis en place pour implémenter la méthode d'analyse pourraient facilement être adaptés à des formes d'annotations plus générales. L'évolution envisagée consisterait donc à étendre notre formalisme d'annotation pour utiliser des profils définis par des notions de sous-typage pouvant accepter des propriétés algébriques plus générales que celle d'Exemption de Motif. Cette dernière serait alors utilisée pour faciliter l'expression de ces sous-types, et ainsi permettre à l'utilisateur de définir des profils exprimant des spécifications beaucoup plus générales que l'approche actuelle. Des approches telles que la notion de sous-typage sémantique [FCB02] montrent en effet qu'une telle évolution resterait compatible avec notre formalisme de sémantique de motif, facilitant ainsi l'adaptation de la méthode d'analyse.

Une autre approche d'évolution possible, en termes d'expressivité, serait de permettre la définition de profils paramétriques. En utilisant des méta-variables dans la définition des profils, à la façon des types polymorphiques dans les langages fonctionnels, il serait ainsi possible de définir des profils prenant arbitrairement certaines propriétés d'Exemption de Motif en entrée et décrivant une spécification du comportement attendu en fonction du motif considéré. Cette approche permettrait l'écriture de profils extrêmement expressifs, mais, contrairement à la proposition précédente, ne permettrait pas d'exprimer des propriétés ne pouvant être décrites par Exemption de Motif.

Ordre Supérieur
Hérité du formalisme de $\lambda-$calcul, les langages de programmation fonctionnels se reposent sur le concept de fonctions d'ordre supérieur, c'est à dire des fonctions pouvant prendre des fonctions en argument et renvoyer des fonctions comme résultat. Parmi les approches de vérification alternatives étudiées, une majorité d'entre elles proposent des solutions permettant l'étude de programmes utilisant ce type de fonctions. C'est en effet une pratique de programmation qui se popularise de plus en plus, au point d'être intégré maintenant dans des langages orientés objet comme Java et C++.

Le formalisme présenté ici étant principalement conçu pour l'étude de transformations de programmes, on a fait le choix de ne considérer que des fonctions du premier ordre. Une évolution naturelle serait donc d'étudier son application à l'étude de programmes utilisant des fonctions d'ordre supérieur. En pratique, encoder de telle fonctions par un système de réécriture nécessite de pouvoir exprimer une application partielle des fonctions. Une solution, proposée dans [Rey68], et utilisée notamment par l'approche de complétion d'automates [Gen14], consiste à introduire un symbole encodant l'application d'une fonction sur son premier paramètre. Il est ainsi possible d'exprimer par des règles de réécriture le comportement d'une telle application (partielle).

La complexité majeure du passage à l'ordre supérieur réside donc dans la définition d'annotations utilisant des profils d'ordre supérieur. De nombreuses solutions, tels que les *Refinement types* [FP91], ont montré que des formalismes similaires d'annotation s'adaptent assez bien au typage de fonctions d'ordre supérieur. On peut notamment citer l'approche proposée par le langage CDuce [BCF03], qui repose sur une notion de typage sémantique assez proche du formalisme de satisfaction de profil proposé ici.

Inférence de profils
Le formalisme d'annotation par Exemption de Motif proposée est, en général, assez peu intrusive et relativement intuitive grâce à sa dépendance à la notion de filtrage par motif. En effet cette approche est largement inspirée de fonctionnalités du monde de la programmation fonctionnelle, ce qui en permet une prise en main assez aisée. Elle nécessite néanmoins un travail d'annotation de la part de l'utilisateur, qui peut sembler important par rapport à certaines solutions alternatives qui proposent des approches beaucoup plus automatisées.

Afin de limiter le travail d'annotation de l'utilisateur, il serait intéressant de proposer une automatisation de la méthode l'analyse basée sur un mécanisme d'inférence des profils. Un tel mécanisme permettrait en théorie, à partir d'un système d'annotations réduits, de déduire l'ensemble des profils nécessaires à la vérification de la spécification algébrique minimale désirée par l'utilisateur. De nombreux scénarios utilisés comme cas d'étude présentent en effet un certain nombre de profils qui pourraient facilement être déduits par une telle approche d'inférence.

Une question importante liée à une telle évolution est la façon dont un tel mécanisme d'inférence s'intègrerait aux autres évolutions proposées. Par exemple, les approches considérées pour améliorer l'expressivité du formalisme serait d'autant plus intéressantes qu'il serait alors possible d'inférer des profils non-triviaux décrivant des propriétés d'autant plus expressives. Néanmoins, plus le formalisme sera expressif plus la complexité d'une telle approche d'inférence sera importante. Des observations similaires sont également évidentes pour le passage à l'ordre supérieur.

Réécriture stratégique et intégration
Enfin, le concept d'origine ayant inspiré les travaux présentés dans cette thèse est principalement celui proposé par Nanopass, dont le but premier n'est pas spécialement de fournir des garanties de transformation de programmes, mais essentiellement de proposer un outil facilitant leur écriture.

Le formalisme de réécriture choisi, ici, propose de définir les transformations de programmes via une approche algébrique similaire, et on a notamment évoqué la notion de réécriture par stratégie comme une alternative à l'approche de Nanopass pour faciliter l'écriture des systèmes de réécriture. Pour revenir à une comparaison avec Nanopass, il serait donc intéressant d'étudier comment l'approche proposée s'intègre à ces notions de réécriture stratégique.

En premier approximation, on pourrait donc étudier l'application de la méthode d'analyse sur des transformations utilisant certaines stratégies classiquement utilisées pour le parcours d'arbres, tels que *Bottom-up*, *Top-down* ou *Innermost*. Pour une application plus complète aux concepts de réécriture stratégique, on pourrait utiliser l'encodage présenté dans [CLM15], qui propose d'exprimer les stratégies via un système de réécriture classique. Cet encodage générant des symboles définis supplémentaires, un mécanisme d'inférence de profils, comme mentionné ci-avant, devrait donc permettre de construire les profils de ces symboles, et ainsi d'appliquer la méthode d'analyse au système généré.

En particulier, on a cité des outils tels que Stratego et Tom, qui proposent des langages de stratégie permettant l'expression et l'implémentation de systèmes de réécriture stratégiques. Une perspective majeure d'application des travaux menés serait donc de proposer une intégration de l'approche présentée dans l'un de ces langages.

A

Preuves et lemmes intermédiaires

A.1 Preuves et lemmes du Chapitre 3

Lemme A.1. *Soit un terme $c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}^n$, on a :*

$$[\![c(t_1, \ldots, t_n)]\!] \quad \subseteq \quad \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$$

Démonstration. Soit un terme linéaire $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}$, on procède par induction sur k, le nombre de symboles définis dans t.

On considère $k = 0$, *i.e.* $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, et une valeur $v \in [\![t]\!]$. Par définition de la sémantique close, il existe une substitution valeur ς telle que $v = \varsigma(t) = c(\varsigma(t_1), \ldots, \varsigma(t_n))$. Donc $v = c(v_1, \ldots, v_n)$, avec $v_i = \varsigma(t_i), \forall i \in [1, n]$, *i.e.* $v_i \in [\![t_i]\!], \forall i \in [1, n]$. On a donc bien $v \in \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$.

On suppose maintenant $k > 0$ tel que $\forall u = c(u_1, \ldots, u_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ ayant moins de k symboles définis, $[\![u]\!] \subseteq \{c(w_1, \ldots, w_n) \mid (w_1, \ldots, w_n) \in [\![u_1]\!] \times \cdots \times [\![u_n]\!]\}$. On considère $v \in [\![t]\!]$; par définition, il existe une position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(p_1, \ldots, p_m)$ avec $\varphi \in \mathcal{D}$, et une valeur $w \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(p_1, \ldots, p_m)$ telles que $v \in [\![t\,[w]_\omega]\!]$. Comme $t(\epsilon) = c \in \mathcal{C}$, $\omega \neq \epsilon$ et on peut donc noter $\omega = i.\omega'$. Ainsi, par induction, $v = c(v_1, \ldots, v_n)$ avec $v_j \in [\![t_j]\!], \forall j \neq i$ et $v_i \in [\![t_i\,[w]_{\omega'}]\!]$. De plus, par définition, on a $[\![t_i\,[w]_{\omega'}]\!] \subseteq [\![t_i]\!]$, d'où $v \in \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \cdots \times [\![t_n]\!]\}$.

Ainsi, par induction, l'inclusion est vérifiée pour tout terme de la forme $c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}^n$. □

Lemme 3.11. *Soit $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}$, pour tout $i \in [1, n]$, on a $\forall v \in [\![t_i]\!]$, $\exists\, c(w_1, \ldots, w_n) \in [\![t]\!]$ tel que $w_i = v$.*

Démonstration. Soit $t = c(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ avec $c \in \mathcal{C}$. On procède par induction sur k, le nombre de symboles définis dans t_i. Si $k = 0$, on a $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, donc pour tout $v \in [\![t_i]\!]$ il existe une substitution σ telle que $\sigma(t_i) \in \mathcal{T}(\mathcal{C})$ et $v = \sigma(t_i)$. De plus, comme $\sigma(t_i) \in \mathcal{T}(\mathcal{C})$, par définition de la sémantique close, on a, pour tout $c(w_1, \ldots, w_n) \in [\![\sigma(t)]\!]$ avec $w_i = \sigma(t_i) = v$. Enfin, comme la sémantique close est conservée par substitution valeur, on a également $w \in [\![t]\!]$.

On considère maintenant t_i ayant $k > 0$ tel que pour tout u_i ayant moins de k symboles définis, $\forall v \in [\![u_i]\!]$, $\exists\, c(w_1, \ldots, w_n) \in [\![t_1, \ldots, u_i, \ldots, t_n]\!]$ tel que $w_i = v$. Soit $v \in [\![t_i]\!]$, *i.e.* il existe une position $\omega \in \mathcal{P}os(t_i)$ telle que $t_{i|\omega} = \varphi_s^{!\mathcal{P}}(p_1, \ldots, p_m)$ avec $\varphi \in \mathcal{D}^m$, et une valeur $w \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(p_1, \ldots, p_m)$ telle que $v \in [\![t_i\,[w]_\omega]\!]$. Par induction, on peut donc conclure qu'il existe $c(w_1, \ldots, w_n) \in [\![t\,[w]_{i.\omega}]\!]$ tel que $w_i = v$, et par définition de la sémantique close, $[\![t\,[w]_{i.\omega}]\!] \subseteq [\![t]\!]$, donc $w \in [\![t]\!]$.

Par induction, on a donc bien pour tout $i \in [1,n]$ et $v \in [\![t_i]\!]$, il existe $c(w_1,\dots,w_n) \in [\![t]\!]$ tel que $w_i = v$. □

Lemme 3.13. *Soient un terme $t \in \mathcal{T}(\mathcal{F},\mathcal{X}) \setminus \mathcal{T}(\mathcal{C},\mathcal{X})$ et une valeur $u \in \mathcal{T}(\mathcal{C})$, $u \in [\![t]\!]$ si et seulement si il existe une position $\omega \in \mathcal{S}td(t) = \{\omega \in \mathcal{P}os(t) \mid \forall \omega' < \omega, t(\omega') \in \mathcal{C}\}$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\dots,t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1,\dots,t_n)$, telles que $u \in [\![t\,[v]_\omega]\!]$.*

Démonstration. La preuve de l'implication inverse est immédiate par définition de la sémantique.

On prouve maintenant l'implication directe par induction sur k le nombre de symboles définis dans t. Si $k = 1$, la preuve est immédiate par définition de la sémantique.

On considère maintenant $k > 1$ tel que pour tout terme q ayant strictement moins de k symboles définis une valeur $w \in [\![q]\!]$ si et seulement si il existe une position $\omega \in \mathcal{S}td(q)$ telle que $q_{|\omega} = \varphi_s^{!\mathcal{P}}(q_1,\dots,q_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(q_1,\dots,q_n)$, telles que $u \in [\![q\,[v]_\omega]\!]$. On considère $u \in [\![t]\!]$, par définition de la sémantique, il existe une position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1,\dots,t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, et une valeur $v \in \mathcal{T}_s(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}}(t_1,\dots,t_n)$, telles que $u \in [\![t\,[v]_\omega]\!]$. Si $\omega \in \mathcal{S}td(t)$, la proposition est vérifiée, sinon, on note $q := t\,[v]_\omega$ d'après l'hypothèse d'induction, il existe une position $\omega' \in \mathcal{S}td(q)$ telle que $q_{|\omega'} = \psi_{s'}^{!\mathcal{P}'}(q_1,\dots,q_m)$ avec $\psi_{s'}^{!\mathcal{P}'} \in \mathcal{D}^m$, et une valeur $v' \in \mathcal{T}_{s'}(\mathcal{C})$ exempte de $\triangleright^{\mathcal{P}'}(q_1,\dots,q_n)$, telles que $u \in [\![q\,[v']_{\omega'}]\!]$.

- Si $\omega' < \omega$, il existe $j \in [1,m]$ et ω'' tels que $\omega = \omega'.j.\omega''$. On note $t_{|\omega'} = \psi_{s'}^{!\mathcal{P}'}(u_1,\dots,u_m)$, et on a donc $q_i = u_i$ pour tout $i \neq j$ et $q_j = u_j\,[v']_{\omega''}$. Par définition de la sémantique on a donc $[\![q_j]\!] \subseteq [\![u_j]\!]$, et par conséquent v' est exempt de $\triangleright^{\mathcal{P}'}(u_1,\dots,u_m)$. Enfin, comme $q\,[v']_{\omega'} = t\,[v']_{\omega'}$, on conclue que $w \in [\![t\,[v']_{\omega'}]\!]$.
- Sinon, comme $\omega' \in \mathcal{S}td(t)$, $q\,[v']_{\omega'} = t\,[v']_{\omega'}\,[v]_\omega$ donc $w[\![t\,[v']_{\omega'}\,[v]_\omega]\!] \subseteq [\![t\,[v']_{\omega'}]\!]$.

Par induction, l'implication directe est donc vérifiée. □

Lemme 3.16. *Soit un terme clos $t \in \mathcal{T}(\mathcal{F})$, pour toute position $\omega \in \mathcal{P}os(t)$ et pour tout termes clos $u, v \in \mathcal{T}_s(\mathcal{F})$ avec $t_{|\omega} : s$, si $[\![u]\!] \subseteq [\![v]\!]$ alors $[\![t\,[u]_\omega]\!] \subseteq [\![t\,[v]_\omega]\!]$.*

Démonstration. Soit un terme clos $t \in \mathcal{T}(\mathcal{F})$, on procède induction sur la forme de t. Si $t = c \in \mathcal{C}^0$, la propriété est clairement vérifiée.

On considère maintenant $t = c(t_1,\dots,t_n)$ avec $c \in \mathcal{C}^n$ et $t_1,\dots,t_n$ respectant la propriété inductive. Si $\omega = \epsilon$, la propriété est clairement vérifiée. Sinon, on note $\omega = i.\omega'$ avec $i \in [1,n]$, et par induction on a $[\![t_i\,[u]_{\omega'}]\!] \subseteq [\![t_i\,[v]_{\omega'}]\!]$. Enfin, on peut conclure avec Proposition 3.5 que $[\![t\,[u]_\omega]\!] \subseteq [\![t\,[v]_\omega]\!]$.

Enfin, on considère $t = \varphi_{s'}^{!\mathcal{P}}(t_1,\dots,t_n)$ avec $\varphi \in \mathcal{D}^n$ et $t_1,\dots,t_n$ respectant la propriété inductive. Si $\omega = \epsilon$, la propriété est clairement vérifiée. Sinon, on note $\omega = i.\omega'$ avec $i \in [1,n]$, et par induction on a $[\![t_i\,[u]_{\omega'}]\!] \subseteq [\![t_i\,[v]_{\omega'}]\!]$. De plus, par Proposition 3.5, on a $[\![t\,[u]_\omega]\!] = [\![x_{s'}^{-p_u}]\!]$ avec $p_u := \triangleright^{\mathcal{P}}(t_1,\dots,t_i\,[u]_{\omega'},\dots,t_n)$, et $[\![t\,[u]_\omega]\!] = [\![x_{s'}^{-p_v}]\!]$ avec $p_v := \triangleright^{\mathcal{P}}(t_1,\dots,t_i\,[v]_{\omega'},\dots,t_n)$. On rappelle que $\triangleright^{\mathcal{P}}(t_1,\dots,t_i\,[u]_{\omega'},\dots,t_n) = \sum_{q\in\mathcal{Q}_u} q$ avec $\mathcal{Q}_u = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}\ t.q.\ t_i\,[u]_{\omega'}$ est exempt de $l_i \wedge \forall j \neq i, t_j$ est exempt de $l_j\}$ et $\triangleright^{\mathcal{P}}(t_1,\dots,t_i\,[v]_{\omega'},\dots,t_n) = \sum_{q\in\mathcal{Q}_v} q$ avec $\mathcal{Q}_v = \{r \mid \exists l_1 * \cdots * l_n \mapsto r \in \mathcal{P}\ t.q.\ t_i\,[v]_{\omega'}$ est exempt de $l_i \wedge \forall j \neq i, t_j$ est exempt de $l_j\}$. Comme $[\![t_i\,[u]_{\omega'}]\!] \subseteq [\![t_i\,[v]_{\omega'}]\!]$, Proposition 3.7 garantit que pour tout profil $l_1 * \cdots * l_n \mapsto r \in \mathcal{P}$, si $t_i\,[v]_{\omega'}$ est exempt de l_i alors $t_i\,[u]_{\omega'}$ est exempt de l_i. D'où $\mathcal{Q}_v \subseteq \mathcal{Q}_u$, et par composition de l'Exemption de Motif avec l'opérateur de disjonction de motif $+$, toute valeur exempte de p_u est exempte de p_v, donc $[\![t\,[u]_\omega]\!] = [\![x_{s'}^{-p_u}]\!] \subseteq [\![t\,[u]_\omega]\!] = [\![x_{s'}^{-p_v}]\!]$.

Par induction, on a donc pour tout $t \in \mathcal{T}(\mathcal{F})$, pour toute position $\omega \in \mathcal{P}os(t)$ et pour tout termes clos $u, v \in \mathcal{T}_s(\mathcal{F})$ avec $t_{|\omega} : s$, si $[\![u]\!] \subseteq [\![v]\!]$ alors $[\![t\,[u]_\omega]\!] \subseteq [\![t\,[v]_\omega]\!]$. □

A.2 Preuves et lemmes du Chapitre 4

Lemme 4.4. *Étant donné $G = (V, E)$ un graphe total, pour tout nœud $u \in V$, il existe $v \in [\![u]\!]$ avec $d(v) \leq n$ si et seulement u est instanciable avec $dist(u) \leq n$.*

Démonstration. Soit $G = (V, E)$ un graphe total, et un nœud $u \in V$. On prouve le Lemme par induction sur n.

Pour $n = 0$:

- On suppose qu'il existe $v \in [\![u]\!]$ avec $d(v) = 0$ avec v est une constante. Si u est un motif, alors u est également une constante et est donc bien instanciable avec $dist(u) = 0$. Si u est nœud variable, alors $[\![v]\!] \subseteq [\![u]\!]$, donc par définition du graphe total, v est nœud du graphe et $(u, v) \in E$. Le nœud v est donc un instanciable et $dist(v) = 0$, donc u est également instanciable avec $dist(u) = 0$.
- On suppose que u est instanciable avec $dist(u) = 0$. Si u est un motif, alors u est également une constante et $u \in [\![u]\!]$ est de profondeur 0. Si u est un nœud variable, alors il existe nœud motif $v \in V$ avec $dist(v) = 0$, donc v est une constante et par définition du graphe total, $[\![v]\!] = \{v\} \subseteq [\![u]\!]$.

On considère maintenant $n > 0$ tel que pour tout nœud $w \in V$, il existe $v' \in [\![w]\!]$ avec $d(v') < n$ si et seulement w est instanciable avec $dist(w) < n$.

- On suppose qu'il existe $v \in [\![u]\!]$ avec $d(v) \leq n$. Si $u = c(u_1, \ldots, u_n)$, alors d'après la Proposition 3.8 $v = c(v_1, \ldots, v_n)$ avec $v_i \in [\![u_i]\!]$ pour tout $i \in [1, n]$. Comme chaque $d(v_i) < n$ pour tout $i \in [1, n]$, par définition du graphe total et induction, chaque u_i est nœud instanciable du graphe avec $dist(u_i) < n$ et $(u, u_i) \in E$. Donc u est instanciable avec $dist(u) \leq n$. Si u est un nœud variable $x_s^{-p} \setminus r$, d'après l'Équation 4.4, il existe $c \in \mathcal{C}_s$ et $q = (q_1, \ldots, q_n) \in Q_c(r + p)$ tels que $v \in [\![c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p})]\!]$. $w = c(z_1 s_1^{-p} \setminus q_1, \ldots, z_n s_n^{-p})$ est donc un nœud instanciable du graphe $dist(w) \leq n$ et $(u, w) \in E$. Donc u est instanciable et $dist(u) \leq n$.
- On suppose que le nœud u est instanciable avec $dist(u) \leq n$. Si $u = c(u_1, \ldots, u_n)$, alors, par définition, chaque u_i est nœud instanciable du graphe avec $dist(u_i) < n$ et $(u, u_i) \in E$. Par induction, pour tout $i \in [1, n]$, il existe $v_i \in [\![u_i]\!]$ avec $d(v_i) < n$. Donc, d'après la Proposition 3.8, $c(v_1, \ldots, v_n) \in [\![u]\!]$ est de profondeur $\leq n$. Si u est nœud variable, alors il existe nœud motif $w \in V$ instanciable avec $dist(w) \leq n$ et $(u, w) \in E$. Donc, il existe $v \in [\![w]\!]$ avec $d(v) \leq n$. Et comme, par définition du graphe total, $[\![w]\!] \subseteq [\![u]\!]$, $v \in [\![u]\!]$.

□

On montre la confluence locale du système $\mathfrak{R}$:

Lemme A.2. *Le système $\mathfrak{R}$ est confluent.*

Démonstration. On prouve la confluence locale du système en montrant que chaque paire critique induite des règles de réécriture converge. On remarque que la préservation de la sémantique donnée par la Proposition 5.4, les conditions des règles P3-P7 restent satisfaites quand une de ces règles est appliquée localement. De plus, les seules règles pouvant réduire un terme de la forme $\overline{x}_s^{-\overline{p}} \setminus \overline{t}$ sont M1 et P7, qui convergent directement, et sont en contradiction avec la condition des règles P3-P6.

On a donc les paires critiques suivantes :
$(A1) - (A2)$ (qui converge directement),
$(A1) - (S1)$ et $(A2) - (S1)$ (qui convergent avec $E1$ et $A1/A2$),

$(A1) - (S2)$ et $(A2) - (S2)$ (qui convergent avec $E2$ et $A1/A2$),
$(A1) - (S3)$ et $(A2) - (S3)$ (qui convergent avec $E3$ et $A1/A2$),
$(A1) - (S4)$ et $(A2) - (S4)$ à droite (qui converge avec $A1/A2$),
$(A1) - (S4)$ à gauche (qui converge avec $A1$),
$(A1) - (M3')$ et $(A2) - (M3')$ (qui convergent avec $M5$ et $A1/A2$),
$(A1) - (M6')$ et $(A2) - (M6')$ (qui convergent avec $M2$),
$(A1) - (L2)$ et $(A2) - (L2)$ (qui convergent avec $L1$ et $A1/A2$),
$(E1) - (S1)$ (qui converge avec 2 fois $E1$, $A1/A2$),
$(E1) - (M6')$ (qui converge avec $M5$ et $E1$, 2 fois $M5$),
$(E1) - (M7)$ seulement possible à gauche (qui converge avec $M5$ et $M2$, n fois $E1$, n fois $A1/A2$),
$(E1) - (M8)$ à gauche (qui converge avec $M5$ et $E1$),
$(E1) - (M8)$ à droite (qui converge avec $M2$),
$(E1) - (T3)$ à gauche (qui converge avec $E2$ et $E2$, $E1$),
$(E1) - (T3)$ à droite (qui converge avec $E2$ et $E3$, $E1$),
$(E1) - (T4)$ à gauche (qui converge avec $E2$),
$(E1) - (T4)$ à droite (qui converge avec $E3$),
$(E1) - (P1)$ (qui converge avec $E3$, et $E3$, $E1$, $M5$),
$(E1) - (P2)$ (qui converge avec $E2$, et $E2$, $E1$, $M5$),
$(E1) - (P3)$ (qui converge avec $E2$, et $E1$, $E2$, $M5$),
$(E2) - (E3)$ (qui converge directement),
$(E2) - (S3)$ et $(E3) - (S2)$ (qui convergent avec 2 fois $S2/S3$, $A1/A2$),
$(E2) - (L5)$ (qui converge avec $E2$, $L1$),
$(E3) - (L4)$ (qui converge avec $E3$, $L1$),
$(E3) - (P5)$ (qui converge avec $E3$, $M5$),
$(S1) - (S4)$ (qui converge avec $S1$, $S4$ et 2 fois $S1$),
$(S1) - (M6')$ (qui converge avec $M3'$, 2 fois $M6'$ et $S1$, 2 fois $M3'$),
$(S1) - (M7)$ seulement possible à gauche (qui converge avec $M3'$, 2 fois $M7$ et $M3'$, n fois $S1$),
$(S1) - (M8)$ à gauche (qui converge avec $M3'$, 2 fois $M8$ et $S1$),
$(S1) - (M8)$ à droite (qui converge avec $M6'$, 2 fois $M8$),
$(S1) - (T3)$ à gauche (qui converge avec $S2$, 2 fois $T3$ et $S2$, $S1$),
$(S1) - (T3)$ à droite (qui converge avec $S3$, 2 fois $T3$ et $S3$, $S1$),
$(S1) - (T4)$ à gauche (qui converge avec $S2$, 2 fois $T4$, $A1/A2$),
$(S1) - (T4)$ à droite (qui converge avec $S3$, 2 fois $T4$, $A1/A2$),
$(S1) - (P1)$ (qui converge avec $S3$, 2 fois $P1$ et $S1$, $M3'$, $S3$),
$(S1) - (P2)$ (qui converge avec $S2$, 2 fois $P2$ et $S1$, $M3'$, $S2$),
$(S1) - (P3)$ (qui converge avec $S2$, 2 fois $P3$ et $S1$, 2 fois $S2$, $M3'$),
$(S2) - (S3)$ (qui converge avec $S3$ et $S2$),
$(S2) - (S4)$ (qui converge avec $S2$, $S4$ et 2 fois $S2$),
$(S2) - (L5)$ (qui converge avec 2 fois $L5$ et $S2$, $L2$),
$(S3) - (S4)$ (qui converge avec $S3$, $S4$ et 2 fois $S3$),
$(S3) - (P5)$ (qui converge avec 2 fois $P5$ et $S3$, 2 fois $M6'$),
$(S3) - (L4)$ (qui converge avec 2 fois $L4$ et $S3$, $L2$),
$(S4) - (M6')$ (qui converge avec 2 fois $M6'$ et $M6'$)
$(S4) - (L2)$ (qui converge avec 2 fois $L2$ et $M6'$, $S4$)
$(M1) - (M3')$ (qui converge avec 2 fois $M1$, $A1/A2$),
$(M1) - (M5)$ (qui converge directement),
$(M1) - (P6)$ (qui converge avec $P7$, puisque $[\![\overline{x}_s^{-\overline{p}} \setminus (\overline{t} + \overline{y}_s^{-\perp})]\!] \subseteq [\![\overline{x}_s^{-\overline{p}} \setminus \overline{y}_s^{-\perp}]\!] = \emptyset$),
$(M1) - (P7)$ (qui converge directement),

$(M1) - (L3)$ (qui converge avec $M1$, $L1$),
$(M2) - (M3')$ (qui converge avec 2 fois $M1$),
$(M2) - (M5)$ (qui converge directement),
$(M2) - (L3)$ (qui converge avec $M2$),
$(T1) - (T2)$ (qui converge directement),
$(L1) - (L3)$ (qui converge avec $M5$ et $M5$, $L1$),
$(L1) - (L4)$ (qui converge avec $E2$ et $E2$, $L1$),
$(L1) - (L5)$ (qui converge avec $E3$ et $E3$, $L1$),
$(L2) - (L3)$ (qui converge avec $M3'$ et $M3'$, 2 fois $L2$),
$(L2) - (L4)$ (qui converge avec $S2$ et $S2$, 2 fois $L2$),
$(L2) - (L5)$ (qui converge avec $S3$ et $S3$, 2 fois $L2$),
$(L5) - (P5)$ (qui converge avec $P5$, et $L5$, $L3$). □

Lemme 4.11. *Soient un motif quasi-additif t et un motif additif linéaire et régulier p. On a :*

1. *Si t est quasi-symbolique, alors $v := t \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit un motif quasi-symbolique.*
2. *$v := t \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t]\!] = \emptyset$.*
3. *$v := (t \setminus p) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t \setminus p]\!] = \emptyset$.*
4. *$v := (t \times p) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de motifs quasi-symboliques. De plus, si t est linéaire, $v = \bot$ si et seulement si $[\![t \times p]\!] = \emptyset$.*

Démonstration. On prouve le Lemme 4.11.1 par induction sur la forme de t.

Si $t = x_s^{-p}$, aucune règle ne s'applique à t, qui est donc sous forme normale. Si $t = x_s^{-p} \setminus u$ avec u un motif quasi-additif, alors par confluence $v = x_s^{-p} \setminus v' \downarrow_{\mathfrak{R}}$ avec $v' = u \downarrow_{\mathfrak{R}}$, qui est soit $\bot$ soit un motif quasi-additif. Si $v' = \bot$, alors la règle M2 s'applique à $x_s^{-p} \setminus \bot$ et le réduit à x_s^{-p}, qui est en forme normale et un motif quasi-symbolique. Sinon, la seule règle pouvant s'appliquer à $x_s^{-p} \setminus v'$ est P7. Si elle s'applique, elle réduit t à $\bot$ qui est en forme normale, et sinon, $t = x_s^{-p} \setminus v'$ est en forme normale et un motif quasi-symbolique.

Si $t = x \mathbin{@} u$ avec u un motif quasi-symbolique, alors par induction sur t, $v' = u \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit un motif quasi-symbolique pattern. If $v' = \bot$, alors L1 est l'unique pouvant s'appliquer à t, qu'elle réduit à $\bot$. Sinon, $x \mathbin{@} v'$ est en forme normale et un motif quasi-symbolique.

Enfin, si $t = c(t_1, \ldots, t_n)$ avec $t_1, \ldots, t_n$ des motifs quasi-symboliques, alors par induction sur t, soit tous les $v_i = t_i \downarrow_{\mathfrak{R}}$ sont des motifs quasi-symboliques, ou il existe $i \in [1, n]$ tel que $t_i \downarrow_{\mathfrak{R}} = \bot$. Dans le premier cas, on a par confluence $v = c(v_1, \ldots, v_n)$ qui est un motif quasi-symbolique. Sinon, $v = c(t_1, \ldots, t_{i-1}, \bot, t_{i+1}, \ldots, t_n) \downarrow_{\mathfrak{R}} = \bot$, avec la règle E1. □

Démonstration. D'après la Proposition 4.7, on sait que la forme normale d'un motif quasi-additif est un motif quasi-additif. De plus :

- si cette forme normale contient $\bot$ en dessous de la racine alors une règle parmi A1, A2, E1, M2 et L1 s'applique forcément ;
- si cette forme normale contient un symbole + sous un symbole constructeur, alors S1 s'applique forcément ;
- si cette forme normale contient un symbole + en dessous d'un symbole @, alors L2 s'applique forcément.

Ainsi la forme normale d'un motif quasi-symbolique est soit $\bot$ soit une somme de motifs quasi-symboliques.

De plus, si $t \downarrow_{\mathfrak{R}} = \bot$ alors la Proposition 5.4 garantit que $[\![t]\!] = [\![\bot]\!] = \emptyset$. On note $v = t \downarrow_{\mathfrak{R}}$, et de même on a $[\![t]\!] = [\![v]\!]$, on prouve maintenant que si $[\![v]\!] = \emptyset$ alors $v = \bot$. On suppose pour cela que $v \neq \bot$ et on montre par induction que v n'est pas en forme normale.

Si $v = x_s^{-p} \setminus u$ et $[\![v]\!] = \emptyset$, alors la règle P7 s'applique, donc v n'est pas en forme normale. Si $v = v_1 + v_2$, alors $[\![v_1]\!] = [\![v_2]\!] = \emptyset$, donc par induction, soit $v_1 = v_2 = \bot$ et les règles A1/A2 s'appliquent, soit v_1 ou v_2 n'est pas en forme normale. Dans les deux cas, v n'est pas forme normale. Si $v = x \mathbin{@} w$, alors $[\![w]\!] = \emptyset$, d'où par induction, soit $w = \bot$ et la règle L1 s'applique, soit w n'est pas en forme normale. Enfin, si $v = c(v_1, \ldots, v_n)$, alors, par linéarité, il existe $i \in [1, n]$ tel que $[\![v_i]\!] = \emptyset$, et par induction, soit $v_i = \bot$ et la règle E1 s'applique, soit v_i n'est pas en forme normale. Dans les deux cas, v n'est pas en forme normale.

Par conséquent, si $v \neq \bot$ est un motif quasi-additif tel que $[\![v]\!] = \emptyset$, alors v n'est pas en forme normale. Ainsi, si $[\![t]\!] = \emptyset$, alors $t \downarrow_{\mathfrak{R}} = \bot$. □

Démonstration. D'après la Proposition 4.7 et le Lemme 4.11.2, on peut considérer, par confluence, que t and p sont en forme normale. On prouve maintenant, par induction sur la forme de p et t, que $v = (t \setminus p) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif.

Si $p = \bot$, alors la règle M2 s'applique à $t \setminus u$ et le réduit à t, qui est en forme normale. Si $p = x_s^{-\bot}$, alors la règle M1 s'applique à $t \setminus u$ et le réduit à $\bot$.

Si $p = c(p_1, \ldots, p_n)$, on procède par induction sur la forme de t :

- Si $t = \bot$, M5 est l'unique règle s'appliquant à $t \setminus p$ qu'elle réduit à $\bot$.
- Si $t = x_s^{-q}$, P7 est l'unique règle pouvant s'appliquer à $t \setminus p$, qu'elle réduit alors à $v = \bot$. Si, elle ne s'applique pas, on a donc $v = x_s^{-q} \setminus p$ qui est un motif quasi-symbolique.
- Si $t = x_s^{-q} \setminus p'$, comme t et p sont en forme normale, P6 est la seule règle s'appliquant à $t \setminus p$, qu'elle réduit à $x_s^{-q} \setminus (p' + p)$. La règle P7 est alors la seule pouvant s'appliquer, en réduisant $x_s^{-q} \setminus (p' + p)$ à $\bot$. Si elle ne s'applique pas, on a donc $v = x_s^{-q} \setminus (p' + p)$ qui est un motif quasi-symbolique.
- Dans le cas où $t = c'(t_1, \ldots, t_m)$, si $c \neq c'$, la règle M8 s'applique à $t \setminus p$, qu'elle réduit à t. Sinon, la règle M7 s'applique à $t \setminus p$, qu'elle réduit à $\sum_{i \in [1,n]} c(t_1, \ldots, t_i \setminus p_i, \ldots, t_n)$, et par induction sur la forme de p, $\forall i, v_i = (t_i \setminus p_i) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif. Par conséquent, $t \setminus p \Longrightarrow_{\mathfrak{R}}^* \sum_{i \in [1,n]} c(t_1, \ldots, v_i, \ldots, t_n)$. De plus, $w = \sum_{i \in [1,n]} c(t_1, \ldots, v_i, \ldots, t_n)$ est un motif quasi-additif, donc d'après le Lemme 4.11.2, $v = w \downarrow_{\mathfrak{R}}$ est un motif quasi-additif. Ainsi, par confluence, $v = (t \setminus p) \downarrow_{\mathfrak{R}}$.
- Si $t = x \mathbin{@} t'$, L3 est la seule règle s'appliquant à $t \setminus p$, qu'elle réduit à $x \mathbin{@} (t' \setminus p)$. De plus, par induction sur t, $v = (t' \setminus p) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif, et $t \setminus p \Longrightarrow_{\mathfrak{R}}^* x \mathbin{@} v$. Si $v = \bot$, alors L1 s'applique à $x \mathbin{@} v$, qu'elle réduit à $\bot$, sinon $x \mathbin{@} v$ est forme normale.
- Si $t = t_1 + t_2$, M3' est la seule règle s'appliquant à $t \setminus p$, qu'elle réduit à $(t_1 \setminus p) + (t_2 \setminus p)$. De plus, par induction sur t, $v_1 = (t_1 \setminus p) \downarrow_{\mathfrak{R}}$ et $v_2 = (t_2 \setminus p) \downarrow_{\mathfrak{R}}$ sont des motifs quasi-additifs. Par conséquent, $t \setminus p \Longrightarrow_{\mathfrak{R}}^* v_1 + v_2$, et comme $v_1 + v_2$ est un motif quasi-additif, le Lemme 4.11.2 garantit que $v = (v_1 + v_2) \downarrow_{\mathfrak{R}} = (t \setminus p) \downarrow_{\mathfrak{R}}$ est motif quasi-additif.

Enfin, si $p = p_1 + p_2$, on procède par induction sur la forme de t :

- La preuve des cas $t = \bot$, $t = x_s^{-q}$, $t = x_s^{-q} \setminus p'$, $t = x \mathbin{@} t'$ et $t = t_1 + t_2$ est identique à celle ci-dessus.
- Si $t = c(t_1, \ldots, t_n)$, alors la règle M6' s'applique à $t \setminus p$, qu'elle réduit à $(t \setminus p_1) \setminus p_2$. De plus, par induction sur la forme de p, $v' = (t \setminus p_1) \downarrow_{\mathfrak{R}}$ et $v = (v' \setminus p_2) \downarrow_{\mathfrak{R}}$ sont des motifs quasi-additifs, et par confluence, $(t \setminus p) \downarrow_{\mathfrak{R}} = v$.

Par induction, on a donc $v = (t \setminus p)\downarrow_{\mathfrak{R}}$ un motif quasi-additif. De plus, d'après le Lemme 4.11.2, v est soit $\perp$, soit une somme de motifs quasi-symboliques. Et, si t est linéaire, on a $v = \perp$ si et seulement si $[\![t \setminus p]\!] = \emptyset$. □

Démonstration. D'après la Proposition 4.7 et le Lemme 4.11.2, on peut considérer, par confluence, que t et p sont en forme normale. On prouve maintenant, par induction sur la forme de p et t, que $v = (t \times p)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif.

Si $t = \perp$, alors la règle E2 réduit $t \times p$ à $\perp$.

Si $t = x_s^{-\perp}$ ou $t = x_s^{-\perp} \setminus w$, on procède par induction sur la forme de p :

- Si $p = \perp$, alors la règle E3 réduit $t \times p$ à $\perp$.
- Si $p = y_s^{-q}$ et $t = x_s^{-\perp}$, alors la règle T1 réduit $t \times p$ à $x \mathbin{@} y_s^{-q}$, qui est en forme normale. Si $t = x_s^{-\perp} \setminus w$, alors la règle P5 réduit $t \times p$ à $(x_s^{-\perp} \times y_s^{-q}) \setminus w$, qui se réduit à $x \mathbin{@} (y_s^{-q} \setminus w)$, via les règles T1 et L3. La règle P7 est alors la seule pouvant s'appliquer à $y_s^{-p} \setminus w$, qu'elle réduit à $\perp$, d'où, avec la règle L1, $(t \times p)\downarrow_{\mathfrak{R}} = \perp$. Et si elle ne s'applique pas, $y_s^{-p} \setminus w$ est en forme normale.
- Si $p = y_s^{-q} \setminus r$ et $t = x_s^{-\perp}$, alors la règle P4 réduit $t \times p$ à $(x_s^{-\perp} \times y_s^{-q}) \setminus r$, qui est ensuite réduit à $x \mathbin{@} (y_s^{-q} \setminus r)$, via les règles T1 et L3. La règle P7 est alors la seule pouvant s'appliquer à $y_s^{-q} \setminus w$, qu'elle réduit à $\perp$, Et si elle ne s'applique pas, $y_s^{-q} \setminus w$ est en forme normale. Si $t = x_s^{-\perp} \setminus w$, alors la règle P5 réduit à $t \times p$ à $(x_s^{-\perp} \times (y_s^{-q} \setminus r)) \setminus w$, qui se réduit à $x \mathbin{@} (y_s^{-q} \setminus (r + w))$, via les règles P4, T1, deux fois L3 et P6. La règle P7 est alors la seule pouvant s'appliquer à $y_s^{-q} \setminus (r + w)$, qu'elle réduit à $\perp$, d'où, avec la règle L1, $(t \times p)\downarrow_{\mathfrak{R}} = \perp$. Si elle ne s'applique pas, $y_s^{-q} \setminus (r + w)$ est en forme normale.
- Si $p = c(p_1, \ldots, p_n)$ et $t = x_s^{-\perp}$, alors la règle P1 réduit $t \times p$ à $x \mathbin{@} (c(z_{1s_1}^{-\perp} \times p_1, \ldots, z_{ns_n}^{-\perp} \times p_n) \setminus \perp)$. De plus, par induction sur on p, pour tout $i \in [1, n]$, $v_i = (z_{is_i}^{-\perp} \times p_i)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif, d'où $c(v_1, \ldots, v_n)$ est un motif quasi-additif. Par conséquent, avec le Lemme 4.11.2, par confluence (et via la règle M2), $(t \times p)\downarrow_{\mathfrak{R}} = (x \mathbin{@} c(v_1, \ldots, v_n))\downarrow_{\mathfrak{R}}$ est un motif quasi-additif. Si $t = x_s^{-\perp} \setminus w$, alors la règle P5 réduit $t \times p$ à $(x_s^{-\perp} \times p) \setminus w$. On a montré que $v' = (x_s^{-\perp} \times p)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif, et donc, avec le Lemme 4.11.3, par confluence, $(t \times p)\downarrow_{\mathfrak{R}} = (v' \setminus w)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif.
- Si $p = y@r$, alors la règle L5 s'applique à $t \times p$, qu'elle réduit à $y@(t \times r)$. D'où, par induction sur p, $v' = (t \times r)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif, et, par confluence, $(t \times r)\downarrow_{\mathfrak{R}} = (y \mathbin{@} v')\downarrow_{\mathfrak{R}}$, qui est un motif quasi-additif, d'après le Lemme 4.11.2.
- Si $p = p_1 + p_2$, alors la règle S3 réduit $t \times p$ à $t \times p_1 + t \times p_2$. De plus, par induction sur p, $v_1 = (t \times p_1)\downarrow_{\mathfrak{R}}$ et $v_2 = (t \times p_2)\downarrow_{\mathfrak{R}}$ sont des motifs quasi-additifs. D'où, par confluence, $v = (t \times p)\downarrow_{\mathfrak{R}} = (v_1 + v_2)\downarrow_{\mathfrak{R}}$, et comme $v_1 + v_2$ est un motif quasi-additif, d'après le Lemme 4.11.2, v également.

Si $t = c(t_1, \ldots, t_n)$, on procède par induction sur la forme de p :

- Si $p = \perp$, alors la règle E3 réduit $t \times p$ à $\perp$.
- Si $p = y_s^{-q}$, alors la règle P2 réduit $t \times$ à $c(t_1 \times z_{1s_1}^{-q}, \ldots, t_n \times z_{ns_n}^{-q}) \setminus q$. De plus, par induction sur t, pour tout $i \in [1, n]$, $v_i = (t_i \times z_{is_i}^{-q})\downarrow_{\mathfrak{R}}$ est un motif quasi-additif, et donc $c(v_1, \ldots, v_n)$ est un motif quasi-additif. Par conséquent, d'après le Lemme 4.11.3, par confluence, $(t \times p)\downarrow_{\mathfrak{R}} = (c(v_1, \ldots, v_n) \setminus q)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif.
- Si $p = y_s^{-q} \setminus r$, alors la règle P5 réduit $t \times p$ à $(t \times y_s^{-q}) \setminus r$. On a montré que $v' = (t \times y_s^{-p})\downarrow_{\mathfrak{R}}$ est un motif quasi-additif pattern, d'où, avec le Lemme 4.11.3, et par confluence, $(t \times p)\downarrow_{\mathfrak{R}} = (v' \setminus r)\downarrow_{\mathfrak{R}}$ est un motif quasi-additif.

- Si $p = c(p_1, \dots, p_n)$, alors soit $c \neq c'$ et la règle T4 réduit $t \times p$ à $\bot$, soit $c = c'$ et la règle T3 réduit $t \times u$ à $c(t_1 \times p_1, \dots, t_n \times p_1)$. De plus, par induction sur t, pour tout $i \in [1, n]$, $v_i = (t_i \times p_i) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif, d'où $c(v_1, \dots, v_n)$ est un motif quasi-additif. Par conséquent, d'après le Lemme 4.11.2, par confluence, $(t \times p) \downarrow_{\mathfrak{R}} = c(v_1, \dots, v_n) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif.
- Pour les cas inductifs $p = y \mathbin{@} r$ et $p = p_1 + p_2$, on procède comme précédemment.

Si $t = x \mathbin{@} w$, alors la règle L4 réduit $t \times p$ à $x \mathbin{@} (w \times p)$, et par induction sur t, $v' = (w \times p) \downarrow_{\mathfrak{R}}$ est un motif quasi-additif. D'où, par confluence, $v = (t \times p) \downarrow_{\mathfrak{R}} = (y \mathbin{@} v') \downarrow_{\mathfrak{R}}$, et comme $y \mathbin{@} v'$ est un motif quasi-additif, d'après le Lemme 4.11.2, v également.

Enfin, si $t = t_1 + t_2$, alors la règle S2 réduit $t \times p$ à $t_1 \times p + t_2 \times p$. De plus, par induction sur t, $v_1 = (t_1 \times p) \downarrow_{\mathfrak{R}}$ et $v_2 = (t_1 \times p) \downarrow_{\mathfrak{R}}$ sont des motifs quasi-additifs. D'où, par confluence, $v = (t \times p) \downarrow_{\mathfrak{R}} = (v_1 + v_2) \downarrow_{\mathfrak{R}}$, et comme $v_1 + v_2$ est un motif quasi-additif, d'après le Lemme 4.11.2, v également.

Par induction, on a donc $v = (t \times p) \downarrow_{\mathfrak{R}}$ un motif quasi-additif. De plus, d'après le Lemme 4.11.2, v est soit $\bot$, soit une somme de motifs quasi-symboliques. Et, si t est linéaire, on a $v = \bot$ si et seulement si $[\![t \times p]\!] = \emptyset$. □

Lemme 4.14. *Soient une version aliassée τ d'un terme $t \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ de sorte s et un motif additif linéaire et régulier p. On a :*

1. *$v := \tau \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une verion aliassée de t ;*
2. *$v := (\tau \setminus p) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. De plus, si t est linéaire et $v = \tau^1 + \tau^2 + \cdots + \tau^m$, alors pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] \iff \exists\, k \in [1, m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$;*
3. *$v := (\tau \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. De plus, si t est linéaire et $v = \tau^1 + \tau^2 + \cdots + \tau^m$, alors pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $[\![\sigma(t)]\!] \subseteq [\![x_s^{-p}]\!] \iff \exists\, k \in [1, m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$.*

Démonstration. On prouve le Lemme 4.14.1 par induction sur la forme de τ et t.

Si $\tau = x \mathbin{@} u$ et $t = x$, avec u un motif quasi-symbolique, alors par confluence $v = x \mathbin{@} v' \downarrow_{\mathfrak{R}}$ avec $v' = u \downarrow_{\mathfrak{R}}$, qui d'après le Lemme 4.11.1 est soit $\bot$, soit un motif quasi-symbolique. Dans le premier cas, la règle L1 s'applique à $x \mathbin{@} \bot$, qu'elle réduit à $\bot$. Sinon, $v = x \mathbin{@} v'$ est une version aliassée de t.

Si $\tau = c(\tau_1, \dots, \tau_n)$ et $t = c(t_1, \dots, t_n)$, avec $\tau_1, \dots, \tau_n$ des versions aliassées de $t_1, \dots, t_n$, respectivement. Par induction, pour chaque $i \in [1, n]$, $v_i = \tau_i \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une version aliassée de t_i. S'il existe $i \in [1, n]$ tel que $v_i = \bot$, alors par confluence, $t \downarrow_{\mathfrak{R}} = c(t_1, \dots, t_{i-1}, \bot, t_{i+1}, \dots, t_n) \downarrow_{\mathfrak{R}}$, d'où, avec la règle E1, $t \downarrow_{\mathfrak{R}} = \bot$. Sinon $c(v_1, \dots, v_n)$ est en forme normale, et une version aliassée de $c(t_1, \dots, t_n)$. □

Démonstration. D'après le Lemme 4.14.1, la forme normale de τ est soit $\bot$ (auquel cas, avec la règle M5, on a $(\tau \setminus p) \downarrow_{\mathfrak{R}} = \bot$) ou une version aliassée de t. On considère donc que τ et p (dont la forme est preservé, d'après le Lemme 4.7) en forme normale. On prouve ainsi le Lemme 4.14.2 par induction sur la forme de p.

On remarque que si $v = (\tau \setminus p) \downarrow_{\mathfrak{R}}$ est une simple version aliassée de t, on a, d'après la Proposition 5.4, $[\![v]\!] = [\![\tau \setminus p]\!]$. D'où, pour toute substitution σ, on a $[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] \iff [\![\sigma(t)]\!] \subseteq [\![v]\!]$.

Si $p = \bot$, alors la règle M2 s'applique à $\tau \setminus p$ qu'elle réduit à τ, qui est en forme normale. Si $p = x_s^{-\bot}$, alors la règle M1 s'applique à $\tau \setminus p$, qu'elle réduit à $\bot$.

Si $p = c(p_1, \dots, p_n)$, on procède par induction sur la forme de t et τ :

- Si $\tau = x \mathbin{@} r$ et $t = x$, alors la règle L3 s'applique à $\tau \setminus p$, qu'elle réduit à $x \mathbin{@} (r \setminus p)$. D'après le Lemme 4.11.3, la forme normale de $(r \setminus p)$ est soit $\bot$, soit une somme $v^1 + \cdots + v^m$, avec chaque v^k un motif quasi-symbolique. Si $r \setminus p \downarrow_{\mathfrak{R}} = \bot$, alors, par confluence, $(\tau \setminus p) \downarrow_{\mathfrak{R}} = \bot$, via la règle L1. Sinon, par confluence, $(\tau \setminus p) \downarrow_{\mathfrak{R}} = (x \mathbin{@} (v^1 + \cdots + v^m)) \downarrow_{\mathfrak{R}}$, et on peut alors appliquer la règle L2 $m-1$ fois pour réduire $x \mathbin{@} (v^1 + \cdots + v^m)$ à $x \mathbin{@} v^1 + \cdots + x \mathbin{@} v^m$ qui est en forme normale, et une somme de version aliassée de t. De plus, si $r = y_s^{-q}$, alors on a soit $(\tau \setminus p) \downarrow_{\mathfrak{R}} = x \mathbin{@} (y_s^{-q} \setminus p)$ ou $(\tau \setminus p) \downarrow_{\mathfrak{R}} = \bot$, et si $r = y_s^{-q} \setminus w$, alors on a soit $(\tau \setminus p) \downarrow_{\mathfrak{R}} = x \mathbin{@} (y_s^{-q} \setminus (w + p))$ ou $(\tau \setminus p) \downarrow_{\mathfrak{R}} = \bot$.
- Si $\tau = c'(\tau_1, \ldots, \tau_m)$ et $t = c'(t_1, \ldots, t_m)$ avec chaque τ_i une version aliassée de t_i, alors soit $c' \neq c$ et la règle M8 s'applique à $\tau \setminus p$, qu'elle réduit à τ, qui est en forme normale, soit $c = c'$. Dans ce cas, la règle M7 s'applique à $\tau \setminus u$, qu'elle réduit à $\sum_{i \in [1,n]} c(\tau_1, \ldots, \tau_i \setminus p_i, \ldots, \tau_n)$. De plus, par induction sur la forme de p, pour tout $i \in [1, n]$, $v_i = (\tau_i \setminus p_i) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit $v_i^1 + \cdots + v_i^{m_i}$, avec chaque v_i^k une forme aliassée de t_i ; on note $m_i = 0$, si $v_i = \bot$. Ainsi, par confluence, on a $\tau \setminus p \downarrow_{\mathfrak{R}} = \sum_{i \in [1,n]} c(\tau_1, \ldots, v_i, \ldots, \tau_n) \downarrow_{\mathfrak{R}}$. D'où, pour chaque $i \in [1, n]$ tel que $v_i = \bot$, on peut appliquer les règles E1 et A1/A2 pour éliminer le terme $c(\tau_1, \ldots, v_i, \ldots, \tau_n)$ de la somme. Si $v_i = \bot$ pour tout $i \in [1, n]$, on a alors $\tau \setminus p \downarrow_{\mathfrak{R}} = \bot$. Sinon, on peut alors appliquer S1 $(\sum_{i \in [1,n]} m_i) - 1$ fois, et on obtient $(\tau \setminus p) \downarrow_{\mathfrak{R}} = \sum_{i \in [1,n]}^{k \in [1,m_i]} c(\tau_1, \ldots, v_i^k, \ldots, \tau_n)$. De plus, si τ est linéaire et pour tout $x \in \mathcal{V}ar(t), \tau_{@x} = y_s^{-q} \vee \tau_{@x} = y_s^{-q} \setminus w$ on a, par induction, pour toute substitution σ telle que $\sigma(t) = c(\sigma(t_1), \ldots, \sigma(t_n)) \in \mathcal{T}(\mathcal{F})$

$$
\begin{aligned}
&[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] \\
&\quad \iff (\forall j \in [1,n], [\![\sigma(t_j)]\!] \subseteq [\![\tau_j]\!]) \wedge (\exists\, i \in [1,n], [\![\sigma(t_i)]\!] \cap [\![p_i]\!] = \emptyset) \\
&\quad \iff (\forall j \in [1,n], [\![\sigma(t_j)]\!] \subseteq [\![\tau_j]\!]) \wedge (\exists\, i \in [1,n], [\![\sigma(t_i)]\!] \subseteq [\![\tau_i \setminus p_i]\!]) \\
&\quad \iff (\forall j \in [1,n], [\![\sigma(t_j)]\!] \subseteq [\![\tau_j]\!]) \wedge (\exists\, i \in [1,n], \exists\, k \in [1,m_i], [\![\sigma(t_i)]\!] \subseteq [\![v_i^k]\!]) \\
&\quad \iff \exists\, i \in [1,n], \exists\, k \in [1,m_i], [\![\sigma(t)]\!] \subseteq [\![c(\tau_1, \ldots, v_i^k, \ldots, \tau_n)]\!]
\end{aligned}
$$

Enfin, si $p = p_1 + p_2$, on procède par induction sur la forme de t et τ :

- Pour le cas où $\tau = x \mathbin{@} r$, on procède comme précédemment.
- Si $\tau = c(\tau_1, \ldots, \tau_n)$ et $t = c'(t_1, \ldots, t_m)$ avec chaque τ_i une version aliassée de t_i, M6' s'applique à $\tau \setminus p$, qu'elle réduit à $(\tau \setminus p_1) \setminus p_2$. De plus, par induction sur la forme de p, $v' = (\tau \setminus p_1) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. Dans le premier cas, la règle M5 réduit $v' \setminus p_2$ à $\bot$, d'où par confluence $\tau \setminus p \downarrow_{\mathfrak{R}} = \bot$. Dans le deuxième cas, $v' = v_1 + \cdots + v_m$, et on peut appliquer la règle M3' $m - 1$ fois sur $(v_1 + \cdots + v_m) \setminus p_2$, pour le réduire à $v_1 \setminus p_2 + \cdots + v_m \setminus p_2$. De même, par induction, on a, pour tout $i \in [1, n]$, $v_i' = (v_i \setminus p_2) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme $v_i^1 + \cdots + v_i^{m_i}$ de forme aliassées de t ; on note $m_i = 0$, si $v_i' = \bot$. On peut alors éliminer les termes $v_i' = \bot$ de la somme via les règles A1/A2, et par confluence, $v = \tau \setminus p \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. Enfin, si τ est linéaire et pour toute variable $x \in \mathcal{V}ar(t), \tau_{@x} = y_s^{-q} \vee \tau_{@x} = y_s^{-q} \setminus w$, on a, par induction, pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$ $[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] = [\![\tau \setminus p_1]\!] \setminus [\![p_2]\!] \iff \exists\, i \in [1,m], [\![\sigma(t)]\!] \subseteq [\![v_i]\!] \setminus [\![p_2]\!] = [\![v_i \setminus p_2]\!] \iff \exists\, i \in [1,m], \exists\, k \in [1, m_i], [\![\sigma(t)]\!] \subseteq [\![v_i^k]\!]$

Par induction, on a donc $v = (\tau \setminus p) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. De plus, si t est linéaire et $v = \tau^1 + \tau^2 + \cdots + \tau^m$, alors pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $[\![\sigma(t)]\!] \subseteq [\![\tau \setminus p]\!] \iff \exists\, k \in [1,m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$. □

Démonstration. On prouve maintenant par induction sur la forme de t, que pour tout motif additif linéaire et régulier p, $v = (t \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme $\tau^1 + \cdots + \tau^m$ de versions aliassées de t, telle que, si t est linéaire, on a :

- $\forall k \in [1,m], \forall x \in \mathcal{V}ar(t), \tau^k_{@x} = y_{s'}^{-p} \vee v^k_{@x} = y_{s'}^{-p} \setminus u$,
- $\forall \sigma, [\![\sigma(t)]\!] \subseteq [\![t \times x_s^{-p}]\!] \iff \exists\, k \in [1,n], [\![\sigma(t)]\!] \subseteq [\![\sigma(\tau^k)]\!]$

On remarque que si $v = (\tau \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ est une simple version aliassée de t, on a, d'après la Proposition 5.4, $[\![v]\!] = [\![\tau \times x_s^{-p}]\!]$. D'où, pour toute substitution σ, on a $[\![\sigma(t)]\!] \subseteq [\![\tau \times x_s^{-p}]\!] \iff [\![\sigma(t)]\!] \subseteq [\![v]\!]$.

Dans le cas $t = x$, T1 est la seule s'appliquant à $t \times x_s^{-p}$, qu'elle réduit à $x \mathbin{@} x_s^{-p}$, qui est une version aliassée de t.

On considère maintenant le cas où $t = c(t_1, \ldots, t_n)$. Dans ce cas, P1 est la seule règle pouvant s'appliquer à $t \times x_s^{-p}$, qu'elle réduit à $c(t_1 \times z_1 s_1^{-p}, \ldots, t_n \times z_n s_n^{-p}) \setminus p$. De plus, par induction, on a, pour tout $i \in [1,n]$, $v_i = (t_i \times z_i s_i^{-p}) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t_i. Par conséquent, on a $t \times x_s^{-p} \Longrightarrow^*_{\mathfrak{R}} c(v_1, \ldots, v_n) \setminus p$.

- S'il existe i tel que $v_i = \bot$, alors la règle E1 s'applique à $c(v_1, \ldots, v_n) \setminus p$, qu'elle réduit à $\bot \setminus p$, qui peut ensuite être réduit à $\bot$ via la règle M5.
- Sinon, par induction, pour tout i, $v_i = \sum_{k=1}^{m_i} v_i^k$ avec chaque v_i^k une version aliassées de t_i. On peut alors appliquer les règles S1 et M3' $\prod_i m_i - 1$ fois, chacune, pour réduire $c(v_1, \ldots, v_n) \setminus p$ à $\sum c(v_1^{k_1}, \ldots, v_n^{k_n}) \setminus p$. Pour chaque $c(v_1^{k_1}, \ldots, v_n^{k_n}) \setminus p$ de cette somme, $c(v_1^{k_1}, \ldots, v_n^{k_n})$ est une version aliassée de t. On sait donc, d'après le Lemme 4.14.2, que $v_{k_1,\ldots,k_n} = (c(v_1^{k_1}, \ldots, v_n^{k_n})) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit $v^1_{k_1,\ldots,k_n} + \cdots + v^{m_{k_1,\ldots,k_n}}_{k_1,\ldots,k_n}$, avec chaque $v^k_{k_1,\ldots,k_n}$ une version aliassée de t ; on note $m_{k_1,\ldots,k_n} = 0$, si $v_i = \bot$. D'où, pour chaque $(k_1, \ldots, k_n) \in [1,m_1] \times \cdots \times [1,m_n]$ tel que $v_{k_1,\ldots,k_n} = \bot$, on peut appliquer les règles A1/A2, pour éliminer ces termes de la somme. On a donc, par confluence, $(t \times x_s^{-p}) \downarrow_{\mathfrak{R}} = \sum_{(k_1,\ldots,k_n) \in [1,m_1] \times \cdots \times [1,m_n]}^{k \in [1, m_{k_1,\ldots,k_n}]} v^k_{k_1,\ldots,k_n}$. De plus, si t est linéaire, on a, par induction, $\forall (k_1, \ldots, k_n) \in [1,m_1] \times \cdots \times [1,m_n], \forall x \in \mathcal{V}ar(t), c(v_1^{k_1}, \ldots, v_n^{k_n})_{@x} = y_s^{-p} \vee c(v_1^{k_1}, \ldots, v_n^{k_n})_{@x} = y_s^{-p} \setminus u$. D'où, d'après le Lemme 4.14.2, $\forall (k_1, \ldots, k_n) \in [1,m_1] \times \cdots \times [1,m_n], \forall k \in [1, m_{k_1,\ldots,k_n}]$, $\forall x \in \mathcal{V}ar(t), v^k_{k_1,\ldots,k_n @x} = y_s^{-p} \vee v^k_{k_1,\ldots,k_n @x} = y_s^{-p} \setminus u$. De même, on a, pour toute substitution σ telle que $\sigma(t) = c(\sigma(t_1), \ldots, \sigma(t_n)) \in \mathcal{T}(\mathcal{F})$:

$$
\begin{array}{l}
[\![\sigma(t)]\!] \subseteq [\![t \times x_s^{-p}]\!] \\
\quad \iff [\![\sigma(t)]\!] \subseteq [\![c(t_1 \times z_1 s_1^{-p}, \ldots, t_n \times z_n s_n^{-p}) \setminus p]\!] \\
\quad \iff (\forall i \in [1,n], [\![\sigma(p_i)]\!] \subseteq [\![t_i \times z_i s_i^{-p}]\!]) \wedge [\![\sigma(t)]\!] \cap [\![p]\!] = \emptyset \\
\quad \iff (\forall i \in [1,n], \exists\, k_i \in [1,m_i], [\![\sigma(t_i)]\!] \subseteq [\![v_i^{k_i}]\!]) \wedge [\![\sigma(t)]\!] \cap [\![p]\!] = \emptyset \\
\quad \iff (\forall i \in [1,n], \exists\, k_i \in [1,m_i], [\![\sigma(t)]\!] \subseteq [\![c(v_1^{k_1}, \ldots, v_n^{k_n})]\!]) \wedge [\![\sigma(t)]\!] \cap [\![p]\!] = \emptyset \\
\quad \iff \forall i \in [1,n], \exists\, k_i \in [1,m_i], [\![\sigma(t)]\!] \subseteq [\![c(v_1^{k_1}, \ldots, v_n^{k_n}) \setminus p]\!] \\
\quad \iff \forall i \in [1,n], \exists\, k_i \in [1,m_i], \exists\, k \in [1, m_{k_1,\ldots,k_n}], [\![\sigma(t_i)]\!] \subseteq [\![v^k_{k_1,\ldots,k_n}]\!]
\end{array}
$$

Par induction, on a donc $v = (\tau \times x_s^{-p}) \downarrow_{\mathfrak{R}}$ est soit $\bot$, soit une somme de versions aliassées de t. De plus, si t est linéaire et $v = \tau^1 + \tau^2 + \cdots + \tau^m$, alors pour toute substitution σ telle que $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $[\![\sigma(t)]\!] \subseteq [\![\tau \times x_s^{-p}]\!] \iff \exists\, k \in [1,m], [\![\sigma(t)]\!] \subseteq [\![\tau^k]\!]$. □

A.3 Preuves et lemmes du Chapitre 5

Lemme 5.5. *Soient p et q des motifs étendus conjonction-linéaires et linéaires à droite d'une conjonction, si $p \Longrightarrow_{\mathfrak{R}} q$, alors pour toute valeur $v \in \mathcal{T}(\mathcal{C})$, et pour toute substitution σ, on a :*

$$p \overset{\sigma}{\preccurlyeq} v \iff q \overset{\sigma}{\preccurlyeq} v$$

Démonstration. Soient p et q des motifs étendus. On commence par remarquer que, pour une variable fraîche z_s^{-p}, pour tout $v \in \mathcal{T}_s(\mathcal{C})$ exempt de p, pour toute substitution σ, il existe une substitution σ', avec $\sigma(x) = \sigma'(x)$ pour toute variable $x \in \mathcal{D}om(\sigma)$, telle que $z_s^{-p} \overset{\sigma'}{\preccurlyeq} v$. Par simplification, on suppose par la suite que si z_s^{-p} est une variable fraîche, on a toujours $z_s^{-p} \overset{\sigma'}{\preccurlyeq} v$ pour tout $v \in \mathcal{T}_s(\mathcal{C})$ exempt de p. De plus, comme on ne considère que des motifs conjonction-linéaires dont le motif à droite de la conjonction est linéaire, on considère que $p_1 \times p_2 \overset{\sigma}{\preccurlyeq} v \iff p_1 \overset{\sigma}{\preccurlyeq} v \wedge p_2 \preccurlyeq v$.

On prouve par induction sur p que la propriété est monotone :

- Si $p = p_1 + p_2$, on considère q_1 et q_2 tels que pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_i \overset{\sigma}{\preccurlyeq} v \iff q_i \overset{\sigma}{\preccurlyeq} v$, on montre que pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 + p_2 \overset{\sigma}{\preccurlyeq} v \iff p_1 + q_2 \overset{\sigma}{\preccurlyeq} v$:

$$\begin{aligned} p_1 + p_2 \overset{\sigma}{\preccurlyeq} v &\iff p_1 \overset{\sigma}{\preccurlyeq} v \vee p_2 \overset{\sigma}{\preccurlyeq} v \\ &\iff p_1 \overset{\sigma}{\preccurlyeq} v \vee q_2 \overset{\sigma}{\preccurlyeq} v \\ &\iff p_1 + q_2 \overset{\sigma}{\preccurlyeq} v \end{aligned}$$

  De même, pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 + p_2 \overset{\sigma}{\preccurlyeq} v \iff q_1 + p_2 \overset{\sigma}{\preccurlyeq} v$.

- Si $p = p_1 \setminus p_2$, on considère q_1 et q_2 tels que pour tou $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_i \overset{\sigma}{\preccurlyeq} v \iff q_i \overset{\sigma}{\preccurlyeq} v$, on montre que, pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 \setminus p_2 \overset{\sigma}{\preccurlyeq} v \iff q_1 \setminus p_2 \overset{\sigma}{\preccurlyeq} v$:

$$\begin{aligned} q_1 \setminus p_2 \overset{\sigma}{\preccurlyeq} v &\iff p_1 \overset{\sigma}{\preccurlyeq} v \wedge p_2 \not\preccurlyeq v \\ &\iff q_1 \overset{\sigma}{\preccurlyeq} v \wedge p_2 \not\preccurlyeq v \\ &\iff q_1 \setminus p_2 \overset{\sigma}{\preccurlyeq} v \end{aligned}$$

  De même, en remarquant que $p_2 \not\preccurlyeq v \iff q_2 \not\preccurlyeq v$, pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 \setminus p_2 \overset{\sigma}{\preccurlyeq} v \iff p_1 \setminus q_2 \overset{\sigma}{\preccurlyeq} v$.

- Si $p = p_1 \times p_2$, on considère q_1 et q_2 tels que pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_i \overset{\sigma}{\preccurlyeq} v \iff q_i \overset{\sigma}{\preccurlyeq} v$, on montre que, pour toute $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 \times p_2 \overset{\sigma}{\preccurlyeq} v \iff p_1 \times q_2 \overset{\sigma}{\preccurlyeq} v$:

$$\begin{aligned} p_1 \times p_2 \overset{\sigma}{\preccurlyeq} v &\iff p_1 \overset{\sigma}{\preccurlyeq} v \wedge p_2 \overset{\sigma}{\preccurlyeq} v \\ &\iff p_1 \overset{\sigma}{\preccurlyeq} v \wedge q_2 \overset{\sigma}{\preccurlyeq} v \\ &\iff p_1 \times q_2 \overset{\sigma}{\preccurlyeq} v \end{aligned}$$

  De même, pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_1 \times p_2 \overset{\sigma}{\preccurlyeq} v \iff q_1 \times p_2 \overset{\sigma}{\preccurlyeq} v$.

- De même, pour $p = x \mathbin{@} p'$, avec q tel que pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p' \overset{\sigma}{\not\ll} v \iff q \overset{\sigma}{\not\ll} v$, le même raisonnement permet de montrer que pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $x \mathbin{@} p' \overset{\sigma}{\not\ll} v \iff x \mathbin{@} q \overset{\sigma}{\not\ll} v$.
- Si $p = c(p_1, \ldots, p_n)$, on considère $q_1, \ldots, q_n$ tels que, pour tout $v \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $p_i \overset{\sigma}{\not\ll} v \iff q_i \overset{\sigma}{\not\ll} v$, on montre que, pour tout $v_1, \ldots, v_n \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $c(p_1, \ldots, p_n) \overset{\sigma}{\not\ll} c(v_1, \ldots, v_n) \iff c(p_1, \ldots, p_{k-1}, q_k, p_{k+1}, \ldots, p_n) \overset{\sigma}{\not\ll} c(v_1, \ldots, v_n)$:

$$\begin{aligned} c(p_1, \ldots, p_n) \overset{\sigma}{\not\ll} c(v_1, \ldots, v_n) &\iff \wedge_{i=1}^{n} p_i \overset{\sigma}{\not\ll} v_i \\ &\iff \wedge_{i=1}^{k-1} p_i \overset{\sigma}{\not\ll} v_i \ \wedge\ q_k \overset{\sigma}{\not\ll} v_k \ \wedge\ \wedge_{i=k+1}^{n} p_i \overset{\sigma}{\not\ll} v_i \\ &\iff c(p_1, \ldots, p_{k-1}, q_k, p_{k+1}, \ldots, p_n) \overset{\sigma}{\not\ll} c(v_1, \ldots, v_n) \end{aligned}$$

On montre que toutes les règles de $\mathfrak{R}$ préservent le filtrage par substitution :

- Pour la plupart des règles, la preuve est immédiate par définition du filtrage par substitution $\overset{\sigma}{\not\ll}$.

($\Rightarrow \bot$) Pour les règles (E1), (E2), (E3), (M5), (T4) et (L1), la preuve est immédiate en montrant (par contradiction) que pour tout $v \in \mathcal{T}(\mathcal{C}), r \not\ll v$, avec r le membre droit des règles considérées.

(M1) De même, pour la règle (M1), on montre par contradiction que pour tout $u \in \mathcal{T}(\mathcal{C}), \overline{v} \setminus \overline{x}_s^{-\bot} \not\ll u$. On suppose donc qu'il existe $u \in \mathcal{T}(\mathcal{C})$ et une substitution σ telle que $\overline{v} \overset{\sigma}{\not\ll} u$ et $\overline{x}_s^{-\bot} \not\ll u$. Comme on ne considère que des motifs bien sortés, on a $\overline{v} : s$, d'où $u : s$, *i.e.* $\overline{x}_s^{-\bot} \not\ll u$ ce qui est une contradiction. Donc, pour tout $u \in \mathcal{T}(\mathcal{C}), \overline{v} \setminus \overline{x}_s^{-\bot} \not\ll u$.

(M6') Pour la règle (M6'), on montre que, pour tout $u \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus (\overline{v} + \overline{w}) \overset{\sigma}{\not\ll} u \iff (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \overline{v}) \setminus \overline{w} \overset{\sigma}{\not\ll} u$:

$$\begin{aligned} \alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus (\overline{v} + \overline{w}) \overset{\sigma}{\not\ll} u &\iff \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \ \wedge\ \overline{v} + \overline{w} \not\ll u \\ &\iff \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \ \wedge\ \overline{v} \not\ll u \wedge \overline{w} \not\ll u \\ &\iff \alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \overline{v} \overset{\sigma}{\not\ll} u \wedge \overline{w} \not\ll u \\ &\iff (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \overline{v}) \setminus \overline{w} \overset{\sigma}{\not\ll} u \end{aligned}$$

(M7) Pour la règle (M7), on montre que, pour tout $u \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n}) \overset{\sigma}{\not\ll} u \iff \alpha(\overline{v_1} \setminus \overline{t_1}, \ldots, \overline{v_n}) + \cdots + \alpha(\overline{v_1}, \ldots, \overline{v_n} \setminus \overline{t_n}) \overset{\sigma}{\not\ll} u$:

$$\begin{aligned} \alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \alpha(\overline{t_1}, \ldots, \overline{t_n}) \overset{\sigma}{\not\ll} u &\iff \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \wedge \alpha(\overline{t_1}, \ldots, \overline{t_n}) \not\ll u \\ &\iff u = \alpha(u_1, \ldots, u_n) \ \wedge\ \wedge_{i=1}^{n} (\overline{v_i} \overset{\sigma}{\not\ll} u_i) \ \wedge\ (\vee_{j=1}^{n} \overline{t_j} \not\ll u_j) \\ &\iff u = \alpha(u_1, \ldots, u_n) \ \wedge\ \vee_{j=1}^{n} (\wedge_{i=1}^{n} (\overline{v_i} \overset{\sigma}{\not\ll} u_i) \wedge \overline{t_j} \not\ll u_j) \\ &\iff \alpha(\overline{v_1} \setminus \overline{t_1}, \ldots, \overline{v_n}) + \cdots + \alpha(\overline{v_1}, \ldots, \overline{v_n} \setminus \overline{t_n}) \overset{\sigma}{\not\ll} u \end{aligned}$$

(T2) Pour la règle (T2), on montre que, pour tout $u \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $\overline{x}_s^{-\overline{p}} \times \overline{y}_s^{-\overline{q}} \overset{\sigma}{\not\ll} u \iff \overline{x} \mathbin{@} \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u$.

Si $p = q$,

$$\begin{array}{rcl}
\overline{x}_s^{-\overline{p}} \times \overline{y}_s^{-\overline{q}} \overset{\sigma}{\not\ll} u & \iff & \overline{x}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \wedge \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \\
& \iff & \sigma(x) = u \wedge u \ \overline{p}\text{-free} \wedge \sigma(y) = u \wedge u \ \overline{p}\text{-free} \\
& \iff & \sigma(x) = u \wedge u \ \bot\text{-free} \wedge \sigma(y) = u \wedge u \ \overline{p}\text{-free} \\
& \iff & \overline{x}_s^{-\bot} \overset{\sigma}{\not\ll} u \wedge \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \\
& \iff & \overline{x} \mathbin{@} \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u
\end{array}$$

Si $q = \bot$,

$$\begin{array}{rcl}
\overline{x}_s^{-\overline{p}} \times \overline{y}_s^{-\overline{q}} \overset{\sigma}{\not\ll} u & \iff & \overline{x}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \wedge \overline{y}_s^{-\bot} \overset{\sigma}{\not\ll} u \\
& \iff & \sigma(x) = u \wedge u \ \overline{p}\text{-free} \wedge \sigma(y) = u \wedge u \ \bot\text{-free} \\
& \iff & \sigma(x) = u \wedge u \ \bot\text{-free} \wedge \sigma(y) = u \wedge u \ \overline{p}\text{-free} \\
& \iff & \overline{x}_s^{-\bot} \overset{\sigma}{\not\ll} u \wedge \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \\
& \iff & \overline{x} \mathbin{@} \overline{y}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u
\end{array}$$

(P1) Pour la règle (P1), on montre que, pour tout $u \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $\overline{x}_s^{-\overline{p}} \times \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \iff \overline{x} \mathbin{@} (\alpha(z_1 s_1^{-\overline{p}} \times \overline{v_1}, \ldots, z_n s_n^{-\overline{p}} \times \overline{v_n}) \setminus \overline{p}) \overset{\sigma}{\not\ll} u$, avec $z_1, \ldots, z_n$ des variables fraîches :

$$\begin{array}{rcl}
\overline{x}_s^{-\overline{p}} \times \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u & \iff & \overline{x}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \ \wedge \ \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \\
& \iff & \sigma(x) = u \wedge u \ \overline{p}\text{-free} \ \wedge \ u = \alpha(u_1, \ldots, u_n) \\
& & \wedge \ \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i) \\
& \iff & \sigma(x) = u = \alpha(u_1, \ldots, u_n) \wedge \overline{p} \not\preccurlyeq u \\
& & \wedge \ \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i \wedge u_i \ \overline{p}\text{-free}) \\
& \iff & \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i \wedge z_i s_i^{-\overline{p}} \overset{\sigma}{\not\ll} u_i) \\
& & \wedge \sigma(x) = u = \alpha(u_1, \ldots, u_n) \wedge \overline{p} \not\preccurlyeq u \\
& \iff & \wedge_{i=1}^n (z_i s_i^{-\overline{p}} \times \overline{v_i} \overset{\sigma}{\not\ll} u_i) \\
& & \wedge \sigma(x) = u = \alpha(u_1, \ldots, u_n) \wedge \overline{p} \not\preccurlyeq u \\
& \iff & \sigma(x) = u \wedge \overline{p} \not\preccurlyeq u \wedge \alpha(z_1 s_1^{-\overline{p}} \times \overline{v_1}, \ldots, z_n s_n^{-\overline{p}} \times \overline{v_n}) \overset{\sigma}{\not\ll} u \\
& \iff & \sigma(x) = u \wedge \alpha(z_1 s_1^{-\overline{p}} \times \overline{v_1}, \ldots, z_n s_n^{-\overline{p}} \times \overline{v_n}) \setminus \overline{p} \overset{\sigma}{\not\ll} u \\
& \iff & \overline{x} \mathbin{@} (\alpha(z_1 s_1^{-p} \times \overline{v_1}, \ldots, z_m s_m^{-p} \times \overline{v_n}) \setminus \overline{p}) \overset{\sigma}{\not\ll} u
\end{array}$$

(P2) Pour la règle (P2), on montre que, pour tout $u \in \mathcal{T}(\mathcal{C})$ et pour toute substitution σ, on a $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \overline{x}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u \iff \alpha(z_1 s_1^{-\overline{p}} \times \overline{v_1}, \ldots, z_n s_n^{-\overline{p}} \times \overline{v_n}) \setminus \overline{p} \overset{\sigma}{\not\ll} u$, avec $z_1, \ldots, z_n$ des variables fraîches :

$$\begin{array}{rcl}
\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \overline{x}_s^{-\overline{p}} \overset{\sigma}{\not\ll} u & \iff & \alpha(\overline{v_1}, \ldots, \overline{v_n}) \overset{\sigma}{\not\ll} u \ \wedge \ \overline{x}_s^{-\overline{p}} \not\preccurlyeq u \\
& \iff & u = \alpha(u_1, \ldots, u_n) \ \wedge \ \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i) \ \wedge \ u \ p\text{-free} \\
& \iff & u = \alpha(u_1, \ldots, u_n) \ \wedge \ \overline{p} \not\preccurlyeq u \\
& & \wedge \ \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i \wedge u_i \ \overline{p}\text{-free}) \\
& \iff & \wedge_{i=1}^n (\overline{v_i} \overset{\sigma}{\not\ll} u_i \wedge z_i s_i^{-\overline{p}} \overset{\sigma}{\not\ll} u_i) \\
& & \wedge u = \alpha(u_1, \ldots, u_n) \wedge \overline{p} \not\preccurlyeq u \\
& \iff & \wedge_{i=1}^n (z_i s_i^{-\overline{p}} \times \overline{v_i} \overset{\sigma}{\not\ll} u_i) \ \wedge \ u = \alpha(u_1, \ldots, u_n) \wedge \overline{p} \not\preccurlyeq u \\
& \iff & \overline{p} \not\preccurlyeq u \wedge \alpha(z_1 s_1^{-\overline{p}} \times \overline{v_1}, \ldots, z_n s_n^{-\overline{p}} \times \overline{v_n}) \overset{\sigma}{\not\ll} u \\
& \iff & \alpha(z_1 s_1^{-p} \times \overline{v_1}, \ldots, z_m s_m^{-p} \times \overline{v_n}) \setminus \overline{p} \overset{\sigma}{\not\ll} u
\end{array}$$

(P7) Pour la règle (P7), on montre par contradiction que, étant donné un motif $\overline{x}_s^{-\overline{p}}$ tel que $\{\!\![\overline{x}_s^{-\overline{p}}]\!\!\} = \emptyset$, pour tout $u \in \mathcal{T}(\mathcal{C})$, on a $\overline{x}_s^{-\overline{p}} \not\ll u$. On suppose qu'il existe $v \in \mathcal{T}(\mathcal{C})$ et une substitution σ telle que $\overline{x}_s^{-\overline{p}} \overset{\sigma}{\ll} v$, on a alors $v \in [\![\overline{x}_s^{-\overline{p}}]\!] \subseteq \{\!\![\overline{x}_s^{-\overline{p}}]\!\!\}$, ce qui est une contradiction. Donc, pour tout $u \in \mathcal{T}(\mathcal{C}), \overline{x}_s^{-\overline{p}} \not\ll u$.

□

Lemme 5.15. *Soient un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$, pour toute position $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, on a $[\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_c \subseteq [\![t]\!]_c$ avec z_s^{-p} une variable fraîche et $p = \triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$.*

Démonstration. Soient un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ et $\omega \in \mathcal{P}os(t)$ telle que $t_{|\omega} = \varphi_s^{!\mathcal{P}}(t_1, \ldots, t_n)$ avec $\varphi_s^{!\mathcal{P}} \in \mathcal{D}^n$, on note $u := t\left[t_{|\omega} \mapsto z_s^{-p}\right]$ avec z_s^{-p} une variable fraîche et $p := \triangleright_c^{\mathcal{P}}(t_1, \ldots, t_n)$.

Si $u \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, par définition, on a $[\![u]\!]_c = \bigcup_{v \in [\![z_s^{-p}]\!]} [\![t\left[t_{|\omega} \mapsto v\right]]\!] \subseteq [\![t]\!]_c$.

Si u a $k > 0$ symboles définis, on suppose maintenant que pour tout $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X}^a)$ tel que $u' := t'\left[t_{|\omega} \mapsto z_s^{-p}\right]$ a strictement moins de k symboles définis, on a $[\![u']\!]_c \subseteq [\![t']\!]_c$, et on montre que $[\![u]\!]_c \subseteq [\![t]\!]_c$. Soit $w \in [\![u]\!]_c$, par définition, il existe une position $\omega' \in \mathcal{P}os(u)$ avec $u_{|\omega} = \psi_{s'}^{!\mathcal{P}'}(u_1, \ldots, u_m)$ où $\psi_{s'}^{!\mathcal{P}'} \in \mathcal{D}^m$ et une valeur $v\mathcal{T}_{s'}(\mathcal{C})$ exemple de $\triangleright_c^{\mathcal{P}'}(u_1, \ldots, u_m)$ telles que $w \in [\![u\left[u_{|\omega'} \mapsto v\right]]\!]_c$. Par hypothèse d'induction, on a donc $w \in [\![t]\!]_c t_{|\omega'} v$ et v exempt de $\triangleright_c^{\mathcal{P}'}(t'_1, \ldots, t'_m)$, avec $t_{|\omega'} = \psi_{s'}^{!\mathcal{P}'}(t'_1, \ldots, t'_m)$. Donc $w \in [\![t]\!]_c$.

Par induction sur le nombre de symboles définis dans u, on a donc bien $[\![u]\!]_c \subseteq [\![t]\!]_c$. □

Lemme 5.17. *Étant donnés une règle de réécriture $ls \twoheadrightarrow rs$, avec $ls = \varphi_s^{!\mathcal{P}}(ls_1, \ldots, ls_n)$ où $\varphi \in \mathcal{D}^n$, préservant la sémantique substitutive $[\![]\!]_s$, un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et une substitution σ^t tels qu'il existe une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \sigma^t(ls)$, alors on a pour toute substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\varsigma)$:*

$$[\![t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]]\!]_\varsigma \subseteq [\![t]\!]_\varsigma$$

Démonstration. Soient une règle $ls \twoheadrightarrow rs$, avec $ls = \varphi_s^{!\mathcal{P}}(l_1, \ldots, l_n)$ où $\varphi \in \mathcal{D}^n$, préservant la sémantique $[\![]\!]_s$, un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et une substitution σ^t tels que tels qu'il existe une position $\omega \in \mathcal{P}os(t)$ avec $t_{|\omega} = \sigma^t(ls)$. On prouve par induction sur k le nombre de symboles définis dans $t_{|\omega}$ que, pour toute substitution valeur ς avec $\mathcal{V}ar(t) \subseteq \mathcal{D}om(\sigma)$, on a $[\![t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]]\!]_\varsigma \subseteq [\![t]\!]_\varsigma$.

- Si $k = 1$, alors φ est le seul symbole défini dans $t_{|\omega}$ donc $\sigma(l_i) \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ pour tout i, et par définition de la sémantique modulo ς, on a $[\![t]\!]_\varsigma = \bigcup_{\varsigma'} [\![t\left[t_{|\omega} \mapsto z_s^{-p}\right]]\!]_{\varsigma'}$ avec ς' une substitution valeur telle que $\varsigma'(z_s^{-p})$ est une valeur exempte de $p := \triangleright^{\mathcal{P}}\left(\varsigma(\sigma^t(l_1)), \ldots, \varsigma(\sigma^t(l_n))\right)$ et $\varsigma'(x) = \varsigma(x)$ pour tout $x \in \mathcal{D}om(\varsigma)$.

  On note $u := t\left[t_{|\omega} \mapsto z_s^{-p}\right]$ et $\sigma := \{z_s^{-p} \mapsto \sigma^t(rs)\}$, on a alors $\sigma(u) = t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]$ et d'après le Lemme 5.12, on a $[\![\sigma(u)]\!]_\varsigma \subseteq \bigcup_{\varsigma'} [\![u]\!]_{\varsigma'}$ avec ς' une substitution valeur telle que $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$ et $\varsigma'(z_s^{-p}) \in [\![\sigma^t(rs)]\!]_\varsigma$.

  Comme $[\![\sigma^t(rs)]\!]_\varsigma \subseteq [\![\sigma^t(ls)]\!]_\varsigma = [\![x_s^{-p}]\!]$, on peut conclure que $[\![t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]]\!]_\varsigma \subseteq [\![t]\!]_\varsigma$.

- On considère maintenant un terme t et une subsitution σ^t tels que $k > 1$ et on suppose que pour tout terme $r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ et toute substitution valeur σ^r tels qu'il existe une position $\omega^r \in \mathcal{P}os(r)$ avec $r_{|\omega} = \sigma^r(ls)$ ayant strictement moins de k symboles définis, on

a, pour toute substitution valeur ς avec $\mathcal{V}ar(r) \subseteq \mathcal{D}om(\varsigma)$, $[\![r\,[\sigma^r(ls) \mapsto \sigma^r(rs)]]\!]_\varsigma \subseteq [\![r]\!]_\varsigma$. Comme $k > 1$, il existe une position $\omega' \in \mathcal{P}os(t)$, avec $\omega < \omega'$ et $t_{|\omega'} = \psi^{!\mathcal{P}'}_{s'}(t_1, \ldots, t_m)$ avec $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ pour tout $i \in [1, n]$, telle que $[\![t]\!]_\varsigma = \bigcup_{\varsigma'} [\![t\left[t_{|\omega'} \mapsto z^{-p}_{s'}\right]]\!]_{\varsigma'}$ avec ς' une substitution valeur telle que $\varsigma'(z^{-p}_{s'})$ est une valeur exempte de $p := \triangleright^{\mathcal{P}'}(\varsigma(t_1), \ldots, \varsigma(t_m))$ et $\varsigma'(x) = \varsigma(x)$ pour tout $x \in \mathcal{D}om(\varsigma)$.

On note $u := t\left[t_{|\omega'} \mapsto z^{-p}_{s'}\right]$ et $\sigma^u := \{x \mapsto \sigma(x)\left[t_{|\omega'} \mapsto z^{-p}_{s'}\right]\}$, et par construction on a $u_{|\omega} = \sigma^u(ls)$, qui a strictement moins de k symboles définis. Donc, par hypothèse d'induction, on a $[\![u\,[\sigma^u(ls) \mapsto \sigma^u(rs)]]\!]_{\varsigma'} \subseteq [\![u]\!]_{\varsigma'}$. De plus, en notant $\sigma := \{z^{-p}_{s'} \mapsto t_{|\omega'}\}$, on a $\sigma(u\,[\sigma^u(ls) \mapsto \sigma^u(rs)]) = t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]$ et d'après le Lemme 5.12, on a $[\![\sigma(u\,[\sigma^u(ls) \mapsto \sigma^u(rs)])]\!]_\varsigma \subseteq \bigcup_{\varsigma'} [\![u\,[\sigma^u(ls) \mapsto \sigma^u(rs)]]\!]_{\varsigma'}$ avec ς' une substitution valeur telle que $\varsigma'(x) = \varsigma(x)$ pour toute variable $x \in \mathcal{D}om(\varsigma)$ et $\varsigma'(z^{-p}_{s'}) \in [\![t_{|\omega'}]\!]_\varsigma = [\![x^{-p}_{s'}]\!]$.

On peut donc conclure que $[\![t\left[\sigma^t(ls) \mapsto \sigma^t(rs)\right]]\!]_\varsigma \subseteq [\![t]\!]_\varsigma$.

Par induction, la relation est donc vérifiée. □

B

Méta-encodage de $\mathfrak{R}$

On donne ci-dessous un méta-encodage du système de règle $\mathfrak{R}$, présenté en Figure 4.8. Des outils de vérification automatique de la terminaison, comme TTT2 ou AProVE ont été utilisés pour montrer la terminaison de l'encodage, et donc du système $\mathfrak{R}$.

La plupart des règles du méta-encodage sont des traductions directes des règles du système $\mathfrak{R}$, en ignorant les conditions dépendantes de l'algorithme `getReachable`. L'encodage des règles M7, T3, P1 et P2 utilisent des symboles intermédiaires pour générer les différents sous-termes du membre droit de la règle. Enfin, l'encodage combine les règles E1 et S1 qui necessitent d'examiner les sous-termes du motif considéré pour déterminer si la règle peut être appliquée. De plus, pour éviter d'examiner infiniment les sous-termes, après l'application maximale de ces règles, les termes générés sont localement gelés jusqu'à l'application d'une autre règle.

```
(VAR u u1 u2 v v1 v2 w f g lu lv n m i p q)
(RULES

  plus(bot,v) -> v
  plus(v,bot) -> v

  times(bot,v) -> bot
  times(v,bot) -> bot


  appl(f,lv) -> split(f,lv,nil)
    split(f,cons(u,lu),lv) -> split(f,lu,cons(u,lv))
    split(f,cons(bot,lu),lv) -> bot
    split(f,cons(plus(u1,u2),lu),lv) -> plus(appl(f,rest(lu,cons(u1,lv))),
                                             appl(f,rest(lu,cons(u2,lv))))
    split(f,nil,lv) -> frozen(f,rest(nil,lv))
    rest(lu,nil) -> lu
    rest(lu,cons(u,lv)) -> rest(cons(u,lu),lv)

  plus(u, plus(v, w)) -> plus(plus(u, v), w)

  times(plus(u1,u2),v) -> plus(times(u1,v),times(u2,v))
  times(v,plus(u1,u2)) -> plus(times(v,u1),times(v,u2))
```

```
minus(v, var(n,bot)) -> bot
minus(v, bot) -> v
minus(plus(v1,v2), w) -> plus(minus(v1, w), minus(v2, w))

minus(bot, v) -> bot

minus(appl(f,lu), plus(v,w)) -> minus(minus(appl(f,lu),v),w)

minus(frozen(f,lu), plus(v,w)) -> minus(minus(appl(f,lu),v),w)

minus(appl(f,lu), appl(f,lv)) -> genm7(f,lu,lv,len(lu))
  genm7(f,lu,lv,z) -> bot
  genm7(f,lu,lv,s(i)) -> plus(genm7(f,lu,lv,i), appl(f,diff(lu,lv,s(i))))
  diff(nil,nil,i) -> nil
  diff(cons(u,lu),cons(v,lv),s(s(i))) -> cons(u,diff(lu,lv,s(i)))
  diff(cons(u,lu),cons(v,lv),s(z)) -> cons(minus(u,v),lu)
  len(nil) -> z
  len(cons(u,lu)) -> s(len(lu))

minus(frozen(f,lu), appl(f,lv)) -> genm7(f,lu,lv,len(lu))
minus(appl(f,lu), frozen(f,lv)) -> genm7(f,lu,lv,len(lu))
minus(frozen(f,lu), frozen(f,lv)) -> genm7(f,lu,lv,len(lu))

minus(appl(f,lu), appl(g,lv)) -> appl(f,lu)

minus(frozen(f,lu), appl(g,lv)) -> appl(f,lu)
minus(appl(f,lu), frozen(g,lv)) -> appl(f,lu)
minus(frozen(f,lu), frozen(g,lv)) -> appl(f,lu)


times(var(m,p), var(n,p)) -> alias(m, var(n, p))
times(var(m,p), var(n,bot)) -> alias(m, var(n, p))
times(var(m,bot), var(n,p)) -> alias(m, var(n, p))

times(appl(f,lu), appl(f,lv)) -> appl(f, genprod(lu,lv))
  genprod(nil, nil) -> nil
  genprod(cons(u,lu), cons(v,lv)) -> cons(times(u,v), genprod(lu,lv))

times(appl(f,lu), frozen(f,lv)) -> appl(f, genprod(lu,lv))
times(frozen(f,lu), appl(f,lv)) -> appl(f, genprod(lu,lv))
times(frozen(f,lu), frozen(f,lv)) -> appl(f, genprod(lu,lv))

times(appl(f,lu), appl(g,lv)) -> bot

times(frozen(f,lu), appl(g,lv)) -> bot
times(appl(f,lu), frozen(g,lv)) -> bot
times(frozen(f,lu), frozen(g,lv)) -> bot
```

```
  alias(n, bot) -> bot
  alias(n, plus(u1,u2)) -> plus(alias(n, u1), alias(n, u2))

  minus(alias(n, v), u) -> alias(n, minus(v,u))
  times(alias(n, v), u) -> alias(n, times(v,u))
  times(v, alias(n, u)) -> alias(n, times(v,u))


  times(var(n,p), appl(f,lv)) -> alias(n,minus(appl(f,genprodl(p, lv)),p))
    genprodl(p, nil) -> nil
    genprodl(p, cons(v, lv)) -> cons(times(var(z, p), v), genprodl(p, lv))

  times(var(n,p), frozen(f,lv)) -> alias(n,minus(appl(f,genprodl(p, lv)),p))

  times(appl(f,lv), var(n,p)) -> minus(appl(f,genprodr(p, lv)),p)
    genprodr(p, nil) -> nil
    genprodr(p, cons(v, lv)) -> cons(times(v, var(z, p)), genprodr(p, lv))

  times(frozen(f,lv), var(n,p)) -> minus(appl(f,genprodr(p, lv)),p)

  times(appl(f,lu), minus(var(m,p), v)) -> minus(times(appl(f,lu), var(m,p)), v)

  times(frozen(f,lu), minus(var(m,p), v)) -> minus(times(appl(f,lu), var(m,p)), v)

  times(var(n,p), minus(var(m,q), v)) -> minus(times(var(n,p), var(m,q)), v)

  times(minus(var(n,p),u), v) -> minus(times(var(n,p),u), v)
  minus(minus(var(n,p),u), v) -> minus(var(n,p),plus(u,v))

  minus(var(n,p),v) -> bot
)
```

C

Cas d'étude

On définit des cas d'études utilisés comme exemple dans le mémoire et/ou disponibles comme fichier de test dans l'implémentation [CL20].

C.1 Aplatissement de liste

On considère des expressions pouvant être un entier ou une liste de telles expressions. Ce langage d'expression peut être représenté par l'algèbre de termes définie par la signature $\Sigma_{list} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} \mathsf{Expr} & := & \mathit{int}(\mathsf{Int}) & \mathsf{Int} & := & z & \mathsf{List} & := & \mathit{nil} \\ & | & \mathit{lst}(\mathsf{List}) & & | & s(\mathsf{Int}) & & | & \mathit{cons}(\mathsf{Expr}, \mathsf{List}) \end{array}$$

Le but est de proposer une fonction d'aplatissement $\mathit{flatten} : \mathsf{List} \mapsto \mathsf{List}$. Le symbole $\mathit{flatten}^{!\mathcal{P}_1}$ est donc annoté avec $\mathcal{P}_1 = \{\bot \mapsto p_{flat}\}$ où $p_{flat} = \mathit{cons}(\mathit{lst}(l_1), l_2)$.

On propose plusieurs versions.

C.1.1 flatten1

Une première version consiste simplement à utiliser une fonction de concaténation de liste *concat*. On a donc les symboles définis suivant :

$$\begin{array}{rcl} \mathit{flatten}^{!\mathcal{P}_1} & : & \mathsf{List} \mapsto \mathsf{List} \\ \mathit{concat}^{!\mathcal{P}_2} & : & \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \end{array}$$

avec $\mathcal{P}_2 = \emptyset$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} \mathit{flatten}(\mathit{nil}) & \to & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \to & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \to & \mathit{flatten}(\mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \to & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \to & l \end{array} \right.$$

C.1.2 flatten2

Une deuxième version consiste à considérer la même fonction *concat* en observant que la concaténation de deux listes plates est une liste plate. On a donc les symboles définis suivant :

$$\begin{array}{lcl} \mathit{flatten}^{!\mathcal{P}_1} & : & \mathsf{List} \mapsto \mathsf{List} \\ \mathit{concat}^{!\mathcal{P}_2} & : & \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \end{array}$$

avec $\mathcal{P}_2 = \{p_{flat} * p_{flat} \mapsto p_{flat}\}$ [1].

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} \mathit{flatten}(\mathit{nil}) & \rightarrow & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \rightarrow & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \rightarrow & \mathit{concat}(\mathit{flatten}(l), \mathit{flatten}(l')) \\ \mathit{concat}(\mathit{cons}(e, l), l') & \rightarrow & \mathit{cons}(e, \mathit{concat}(l, l')) \\ \mathit{concat}(\mathit{nil}, l) & \rightarrow & l \end{array} \right.$$

C.1.3 flatten3

Une dernière version consiste à considérer une fonction *fconcat* qui effectue à la fois une concaténation de deux listes, et leur aplatissement. On a donc les symboles définis suivant :

$$\begin{array}{lcl} \mathit{flatten}^{!\mathcal{P}_1} & : & \mathsf{List} \mapsto \mathsf{List} \\ \mathit{fconcat}^{!\mathcal{P}_2} & : & \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \end{array}$$

avec $\mathcal{P}_2 = \{\bot * \bot \mapsto p_{flat}\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} \mathit{flatten}(\mathit{nil}) & \rightarrow & \mathit{nil} \\ \mathit{flatten}(\mathit{cons}(\mathit{int}(n), l)) & \rightarrow & \mathit{cons}(\mathit{int}(n), \mathit{flatten}(l)) \\ \mathit{flatten}(\mathit{cons}(\mathit{lst}(l), l')) & \rightarrow & \mathit{fconcat}(l, l') \\ \mathit{fconcat}(\mathit{cons}(\mathit{int}(n), l), l') & \rightarrow & \mathit{cons}(\mathit{int}(n), \mathit{fconcat}(l, l')) \\ \mathit{fconcat}(\mathit{cons}(\mathit{lst}(l_1), l_2), l) & \rightarrow & \mathit{fconcat}(l_1, \mathit{cons}(\mathit{lst}(l_2), l)) \\ \mathit{fconcat}(\mathit{nil}, l) & \rightarrow & \mathit{flatten}(l) \end{array} \right.$$

C.2 Formule logique

On considère des formules logiques représentées par les termes de l'algèbre définie par la signature $\Sigma_{formula} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{lcl} \mathsf{Int} & := & z \\ & | & s(\mathsf{Int}) \\ \\ \mathsf{Var} & := & \mathit{var}(\mathsf{Int}) \\ & | & \mathit{skolem}(\mathsf{VarList}) \\ \\ \mathsf{VarList} & := & \mathit{nil} \\ & | & \mathit{cons}(\mathsf{Var}, \mathsf{VarList}) \end{array}$$

$$\begin{array}{lcl} \mathsf{Formula} & := & \mathit{predicate}(\mathsf{Int}, \mathsf{VarList}) \\ & | & \mathit{not}(\mathsf{Formula}) \\ & | & \mathit{and}(\mathsf{Formula}, \mathsf{Formula}) \\ & | & \mathit{or}(\mathsf{Formula}, \mathsf{Formula}) \\ & | & \mathit{impl}(\mathsf{Formula}, \mathsf{Formula}) \\ & | & \mathit{exists}(\mathsf{Var}, \mathsf{Formula}) \\ & | & \mathit{forall}(\mathsf{Var}, \mathsf{Formula}) \end{array}$$

On considère différentes transformations classiques des formules logiques.

1. le système ne préserve pas la sémantique sans cette annotation, comme défini et vérifié dans l'exemple flatten_fail fourni avec l'implémentation

C.2.1 Forme normale négative

On considère une fonction *nnf* qui transforme une formule logique en sa forme normale négative. On a donc le symbole défini :

$$nnf^{!\mathcal{P}} : \mathsf{Formula} \mapsto \mathsf{Formula}$$

avec $\mathcal{P} = \{\bot \mapsto impl(f_1, f_2) + not(!predicate(i, l))\}$.

Le comportement de la fonction associée à ces symboles est décrit par le CBTRS suivant :

$$\begin{cases}
nnf(predicate(s,t)) & \rightarrow & predicate(s,t) \\
nnf(not(predicate(s,t))) & \rightarrow & not(predicate(s,t)) \\
nnf(not(and(p1,p2))) & \rightarrow & or(nnf(not(p1)), nnf(not(p2))) \\
nnf(not(or(p1,p2))) & \rightarrow & and(nnf(not(p1)), nnf(not(p2))) \\
nnf(not(exists(s,p))) & \rightarrow & forall(s, nnf(not(p))) \\
nnf(not(forall(s,p))) & \rightarrow & exists(s, nnf(not(p))) \\
nnf(not(not(p))) & \rightarrow & nnf(p) \\
nnf(not(impl(p1,p2))) & \rightarrow & and(nnf(p1), nnf(not(p2))) \\
nnf(impl(p1,p2)) & \rightarrow & or(nnf(not(p1)), nnf(p2)) \\
nnf(and(p1,p2)) & \rightarrow & and(nnf(p1), nnf(p2)) \\
nnf(or(p1,p2)) & \rightarrow & or(nnf(p1), nnf(p2)) \\
nnf(exists(s,p)) & \rightarrow & exists(s, nnf(p)) \\
nnf(forall(s,p)) & \rightarrow & forall(s, nnf(p))
\end{cases}$$

C.2.2 Skolémisation

On considère une fonction *skolemize* de skolémisation des formules logiques. Pour effectuer la skolémisation, on introduit également des fonctions *replaceVar* et *filterVar* qui remplace la variable skolémisée dans une formule, et dans une liste de variables, respectivement. On a donc les symboles définis suivant :

$$\begin{array}{lcl}
skolemize^{!\mathcal{P}} & : & \mathsf{Formula} * \mathsf{VarList} \mapsto \mathsf{Formula} \\
replaceVar & : & \mathsf{Formula} * \mathsf{Var} \mapsto \mathsf{Formula} \\
filterVar & : & \mathsf{VarList} * \mathsf{Var} \mapsto \mathsf{VarList}
\end{array}$$

avec $\mathcal{P} = \{\bot \mapsto exists(x, y)\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\begin{cases}
skolemize(predicate(s, xs), l) & \rightarrow & predicate(s, xs) \\
skolemize(and(p_1, p_2), l) & \rightarrow & and(skolemize(p_1, l), skolemize(p_2, l)) \\
skolemize(or(p_1, p_2), l) & \rightarrow & or(skolemize(p_1, l), skolemize(p_2, l)) \\
skolemize(exists(x, p), l) & \rightarrow & skolemize(replaceVar(p, skolem(x, l))) \\
skolemize(forall(x, p), l) & \rightarrow & forall(x, skolemize(p, cons(var(x), l))) \\
replaceVar(predicate(s, xs), skl) & \rightarrow & predicate(s, replaceVar(xs, skl)) \\
replaceVar(and(p_1, p_2), skl) & \rightarrow & and(replaceVar(p_1, skl), replaceVar(p_2, skl)) \\
replaceVar(or(p_1, p_2), skl) & \rightarrow & or(replaceVar(p_1, skl), replaceVar(p_2, skl)) \\
replaceVar(exists(x, p), skl) & \rightarrow & exists(x, replaceVar(p, skl)) \\
replaceVar(forall(x, p), skl) & \rightarrow & forall(x, replaceVar(p, skl)) \\
filterVar(cons(x, xs), skolem(x, l)) & \rightarrow & cons(skolem(x, l), filterVar(xs, skolem(x, l))) \\
filterVar(cons(x, xs), skolem(x', l)) & \rightarrow & cons(x, filterVar(xs, skolem(x', l))) \\
filterVar(nil, skl) & \rightarrow & nil
\end{cases}$$

C.3 Algèbre de Peano

On considère des entiers naturels représentés par des termes de l'algèbre de Peano définie par la signature de type suivante :

$$\begin{array}{rcl} \mathsf{Int} & := & z \\ & | & s(\mathsf{Int}) \end{array}$$

On considère différentes transformations dans l'algèbre de Peano.

C.3.1 Simplification d'addition

On considère un langage d'expression définie dans l'algèbre de Peano par des additions d'entiers ou de variables. Ces expressions peuvent donc être décrits par des termes de sorte Expr, décrite par la signature :

$$\begin{array}{rcl} \mathsf{Expr} & := & \mathit{int}(\mathsf{Int}) \\ & | & \mathit{var}(\mathsf{Int}) \\ & | & \mathit{plus}(\mathsf{Expr}, \mathsf{Expr}) \end{array}$$

On considère dans ce langage d'expression, une fonction *removePlus0* qui simplifie une expression en éliminant l'élément neutre de l'addition, *i.e.* l'entier 0 représenté par le constructeur z. On a donc le symbole défini :

$$\mathit{removePlus0}^{!\mathcal{P}_1} \quad : \quad \mathsf{Expr} \mapsto \mathsf{Expr}$$

avec $\mathcal{P}_1 = \{\bot \mapsto \mathit{plus}(\mathit{int}(z), e), \bot \mapsto \mathit{plus}(e, \mathit{int}(z))\}$

On peut vérifier que le CBTRS suivant ne préserve pas la sémantique :

$$\left\{ \begin{array}{lcl} \mathit{removePlus0}(\mathit{int}(n)) & \to & \mathit{int}(n) \\ \mathit{removePlus0}(\mathit{var}(n)) & \to & \mathit{var}(n) \\ \mathit{removePlus0}(\mathit{plus}(\mathit{int}(z), e)) & \to & \mathit{removePlus0}(e) \\ \mathit{removePlus0}(\mathit{plus}(e, \mathit{int}(z))) & \to & \mathit{removePlus0}(e) \\ \mathit{removePlus0}(\mathit{plus}(e_1 \setminus \mathit{int}(z), e_2 \setminus \mathit{int}(z))) & \to & \mathit{plus}(\mathit{removePlus0}(e_1), \mathit{removePlus0}(e_2)) \end{array} \right.$$

On considère donc une fonction intermédiaire *add* qui construit une addition simplifiée de deux expressions. On a donc les symboles définis suivant :

$$\begin{array}{rcl} \mathit{removePlus0}^{!\mathcal{P}_1} & : & \mathsf{Expr} \mapsto \mathsf{Expr} \\ \mathit{add}^{!\mathcal{P}_2} & : & \mathsf{Expr} * \mathsf{Expr} \mapsto \mathsf{Expr} \end{array}$$

avec $\mathcal{P}_2 = \{\mathit{plus}(\mathit{int}(z), e) * \mathit{plus}(\mathit{int}(z), e) \mapsto \mathit{plus}(\mathit{int}(z), e), \mathit{plus}(e, \mathit{int}(z)) * \mathit{plus}(e, \mathit{int}(z)) \mapsto \mathit{plus}(e, \mathit{int}(z))\}$

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} \mathit{removePlus0}(\mathit{int}(n)) & \to & \mathit{int}(n) \\ \mathit{removePlus0}(\mathit{var}(n)) & \to & \mathit{var}(n) \\ \mathit{removePlus0}(\mathit{plus}(e_1, e_2)) & \to & \mathit{add}(\mathit{removePlus0}(e_1), \mathit{removePlus0}(e_2)) \\ \mathit{add}(\mathit{int}(z), e) & \to & e \\ \mathit{add}(e, \mathit{int}(z))) & \to & e \\ \mathit{add}(e_1 \setminus \mathit{int}(z), e_2 \setminus \mathit{int}(z)) & \to & \mathit{plus}(e_1, e_2) \end{array} \right.$$

C.3.2 Multiplication par 0

On considère des fonctions d'addition *plus* et de multiplication *mult* et on vérifie qu'une multiplication par 0 vaut toujours 0. On a donc les symboles définis suivant :

$$\begin{array}{rcl} plus^{!\mathcal{P}_1} & : & \mathsf{Int} * \mathsf{Int} \mapsto \mathsf{Int} \\ mult^{!\mathcal{P}_2} & : & \mathsf{Int} * \mathsf{Int} \mapsto \mathsf{Int} \end{array}$$

avec $\mathcal{P}_1 = \{s(i) * s(i) \mapsto s(i)\}$ et $\mathcal{P}_2 = \{s(i) * \bot \mapsto s(i), \bot * s(i) \mapsto s(i)\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} plus(z, n) & \rightarrowtriangle & n \\ plus(s(i), n) & \rightarrowtriangle & s(plus(i, n)) \\ mult(z, n) & \rightarrowtriangle & z \\ mult(s(i), n) & \rightarrowtriangle & plus(n, mult(i, n)) \end{array} \right.$$

C.4 Listes triées

On considère une algèbre de listes abstraites définie par la signature $\Sigma_{sorted} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{rcl} \mathsf{Expr} & := & A \\ & | & B \end{array} \qquad \begin{array}{rcl} \mathsf{List} & := & nil \\ & | & cons(\mathsf{Expr}, \mathsf{List}) \end{array}$$

où les constructeur A et B représentent des valeurs telles que $A < B$.

On peut ainsi représenter les listes triées par les valeurs de sorte List exemptes du motif $p_{sorted} = cons(B, cons(A, l))$.

On considère différentes transformations sur les listes triés.

C.4.1 Tri insertion

On considère une fonction *sort* effectuant un tri par insertion des listes via une fonction *insert*. On utilise également des fonctions *leq* et *ite* qui correspondent à l'opérateur $\leq$ et à la construction *if-then-else*. On a donc les symboles définis suivant :

$$\begin{array}{rcl} sort^{!\mathcal{P}_1} & : & \mathsf{List} \mapsto \mathsf{List} \\ insert^{!\mathcal{P}_2} & : & \mathsf{Expr} * \mathsf{List} \mapsto \mathsf{List} \\ ite^{!\mathcal{P}_{if}} & : & \mathsf{Bool} * \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \\ leq^{!\mathcal{P}_{\leq}} & : & \mathsf{Expr} * \mathsf{Expr} \mapsto \mathsf{Bool} \end{array}$$

avec $\mathcal{P}_1 = \{\bot \mapsto p_{sorted}\}$, $\mathcal{P}_2 = \{\bot * p_{sorted} \mapsto p_{sorted}\}$, $\mathcal{P}_{if} = \{false * p_{sorted} * \bot \mapsto p_{sorted}, true * \bot * p_{sorted} \mapsto p_{sorted}\}$ et $\mathcal{P}_{\leq} = \{\bot * A \mapsto false, B * \bot \mapsto false, A * B \mapsto true\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{ \begin{array}{lcl} leq(x, B) & \rightarrowtriangle & true \\ leq(A, y) & \rightarrowtriangle & true \\ leq(B, A) & \rightarrowtriangle & false \\ ite(true, l, l') & \rightarrowtriangle & l \\ ite(false, l, l') & \rightarrowtriangle & l' \\ insert(x, nil) & \rightarrowtriangle & cons(x, nil) \\ insert(x, cons(y, l)) & \rightarrowtriangle & ite(leq(x, y), cons(x, cons(y, l)), cons(y, insert(x, l))) \\ sort(cons(x, l)) & \rightarrowtriangle & insert(x, sort(l)) \\ sort(nil) & \rightarrowtriangle & nil \end{array} \right.$$

Pour vérifier la préservation de la sémantique avec l'approche par linéarisation, on peut remplacer le membre gauche de la règle

$$insert(x, cons(y, l)) \quad \to\!\!\!\!\triangleright \quad ite(leq(x, y), cons(x, cons(y, l)), cons(y, insert(x, l)))$$

par $insert(x \mathbin{@} (A + B), cons(y \mathbin{@} (A + B), l))$.

C.4.2 Tri fusion

On considère une fonction *sort* effectuant un tri fusion des listes. Pour se faire, on introduit une sorte de paire de listes :

$$\mathsf{Pair} \quad := \quad pair(\mathsf{List}, \mathsf{List})$$

On utilise toujours des fonctions *leq* et *ite* qui correspondent à l'opérateur $\leq$ et à la construction *if-then-else*, ainsi que des *split*, *sortP* et *merge*, qui sépare une liste en une paire, trie une paire de liste et fusionne une paire de listes triées. On a donc les symboles définis suivant :

$$\begin{array}{rcl}
sort^{!\mathcal{P}_1} & : & \mathsf{List} \mapsto \mathsf{List} \\
split & : & \mathsf{List} * \mathsf{List} * \mathsf{List} \mapsto \mathsf{Pair} \\
sortP^{!\mathcal{P}_2} & : & \mathsf{Pair} \mapsto \mathsf{Pair} \\
merge^{!\mathcal{P}_3} & : & \mathsf{Pair} \mapsto \mathsf{List} \\
ite^{!\mathcal{P}_{if}} & : & \mathsf{Bool} * \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \\
leq^{!\mathcal{P}_\leq} & : & \mathsf{Expr} * \mathsf{Expr} \mapsto \mathsf{Bool}
\end{array}$$

avec $\mathcal{P}_1 = \{\bot \mapsto p_{sorted}\}$, $\mathcal{P}_2 = \{\bot \mapsto p_{sorted}\}$, $\mathcal{P}_3 = \{p_{sorted} \mapsto p_{sorted}, cons(A, l)) \mapsto cons(A, l))\}$, $\mathcal{P}_{if} = \{false * p_{sorted} * \bot \mapsto p_{sorted}, true * \bot * p_{sorted} \mapsto p_{sorted}, false * cons(A, l)) * \bot \mapsto cons(A, l)\}$ et $\mathcal{P}_\leq = \{\bot * A \mapsto false, B * \bot \mapsto false, A * B \mapsto true\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\left\{\begin{array}{lcl}
leq(x, B) & \to\!\!\!\!\triangleright & true \\
leq(A, y) & \to\!\!\!\!\triangleright & true \\
leq(B, A) & \to\!\!\!\!\triangleright & false \\
ite(true, l, l') & \to\!\!\!\!\triangleright & l \\
ite(false, l, l') & \to\!\!\!\!\triangleright & l' \\
split(nil, l_1, l_2) & \to\!\!\!\!\triangleright & pair(l_1, l_2) \\
split(cons(x, nil), l_1, l_2) & \to\!\!\!\!\triangleright & pair(cons(x, l_1), l_2) \\
split(cons(x, cons(y, l)), l_1, l_2) & \to\!\!\!\!\triangleright & split(l, cons(x, l_1), cons(y, l_2)) \\
merge(pair(nil, l)) & \to\!\!\!\!\triangleright & l \\
merge(pair(l, nil)) & \to\!\!\!\!\triangleright & l \\
merge(pair(cons(x, l_1), cons(y, l_2))) & \to\!\!\!\!\triangleright & ite(leq(x, y), cons(x, merge(pair(l_1, cons(y, l_2)))), \\
 & & \quad cons(y, merge(pair(cons(x, l_1), l_2)))) \\
sortP(pair(l_1, l_2)) & \to\!\!\!\!\triangleright & pair(sort(l_1), sort(l_2)) \\
sort(l) & \to\!\!\!\!\triangleright & merge(sortP(split(l, nil, nil)))
\end{array}\right.$$

Pour vérifier la préservation de la sémantique avec l'approche par linéarisation, on peut remplacer le membre gauche de la règle

$$merge(pair(cons(x, l_1), cons(y, l_2))) \quad \to\!\!\!\!\triangleright \quad ite(leq(x, y), cons(x, merge(pair(l_1, cons(y, l_2)))), \quad cons(y, merge(pair(cons(x, l_1), l_2))))$$

par $merge(pair(cons(x \mathbin{@} (A + B), l_1), cons(y \mathbin{@} (A + B), l_2)))$.

C.4.3 Suppression

On considère une fonction *delete* supprimant la première occurrence d'un élément dans une liste, et on vérifie qu'une liste reste triée par suppression. On utilise également des fonctions *eq* et *ite* qui correspondent à l'opérateur = et à la construction *if-then-else*. On a donc les symboles définis suivant :

$$\begin{array}{rcl} delete^{!\mathcal{P}} & : & \mathsf{Expr} * \mathsf{List} \mapsto \mathsf{List} \\ ite^{!\mathcal{P}_{if}} & : & \mathsf{Bool} * \mathsf{List} * \mathsf{List} \mapsto \mathsf{List} \\ eq^{!\mathcal{P}_{=}} & : & \mathsf{Expr} * \mathsf{Expr} \mapsto \mathsf{Bool} \end{array}$$

avec $\mathcal{P} = \{\bot * p_{sorted} \mapsto p_{sorted}, \bot * cons(A,l) \mapsto cons(A,l)\}$, $\mathcal{P}_{if} = \{false * p_{sorted} * \bot \mapsto p_{sorted}, true * \bot * p_{sorted} \mapsto p_{sorted}, false * cons(A,l)) * \bot \mapsto cons(A,l), true * \bot * cons(A,l)) \mapsto cons(A,l)\}$ et $\mathcal{P}_{=} = \{A * A \mapsto false, B * B \mapsto false, A * B \mapsto true, B * A \mapsto true\}$.

Le comportement des fonctions associées à ces symboles est décrit par le CBTRS suivant :

$$\begin{cases} eq(A,A) & \twoheadrightarrow & true \\ eq(B,B) & \twoheadrightarrow & true \\ eq(A,B) & \twoheadrightarrow & false \\ eq(B,A) & \twoheadrightarrow & false \\ ite(true,l,l') & \twoheadrightarrow & l \\ ite(false,l,l') & \twoheadrightarrow & l' \\ delete(x,nil) & \twoheadrightarrow & nil \\ delete(x,cons(y,l)) & \twoheadrightarrow & ite(eq(x,y),l,cons(y,delete(x,l))) \end{cases}$$

Pour vérifier la préservation de la sémantique avec l'approche par linéarisation, on peut remplacer le membre gauche de la règle

$$delete(x,cons(y,l)) \twoheadrightarrow ite(eq(x,y),l,cons(y,delete(x,l)))$$

par $delete(x \mathbin{@} (A+B), cons(y \mathbin{@} (A+B), l))$.

On peut également prouver que dans une liste quelconque, la suppression de toutes les occurrences d'un élément donné garantit son absence. On considère alors les annotations $\mathcal{P} = \{A * \bot \mapsto B, B * \bot \mapsto A\}$ et $\mathcal{P}_{if} = \{false * A * \bot \mapsto A, true * \bot * A \mapsto A, false * B * \bot \mapsto B, true * \bot * B \mapsto B\}$.

Et le comportement des fonctions associées à ces symboles est alors décrit par le CBTRS suivant :

$$\begin{cases} eq(A,A) & \twoheadrightarrow & true \\ eq(B,B) & \twoheadrightarrow & true \\ eq(A,B) & \twoheadrightarrow & false \\ eq(B,A) & \twoheadrightarrow & false \\ ite(true,l,l') & \twoheadrightarrow & l \\ ite(false,l,l') & \twoheadrightarrow & l' \\ delete(x,nil) & \twoheadrightarrow & nil \\ delete(x,cons(y,l)) & \twoheadrightarrow & ite(eq(x,y),delete(x,l),cons(y,delete(x,l))) \end{cases}$$

Pour vérifier la préservation de la sémantique avec l'approche par linéarisation, on peut remplacer le membre gauche de la règle

$$delete(x,cons(y,l)) \twoheadrightarrow ite(eq(x,y),delete(x,l),cons(y,delete(x,l)))$$

par $delete(x \mathbin{@} (A+B), cons(y \mathbin{@} (A+B), l))$.

C.4.4 Inversion

On considère une fonction *reverse* d'inversion des listes et on vérifie que l'inversion d'une liste triée est inversement triée On a donc les symboles définis suivant :

$$reverse^{!\mathcal{P}} \quad : \quad \mathsf{List} * \mathsf{List} \mapsto \mathsf{List}$$

avec $\mathcal{P} = \{p_{sorted} * cons(B,l) \mapsto p_{inv}, cons(A,l) * p_{inv} \mapsto p_{inv}\}$ où $p_{inv} = cons(A, cons(B,l))$.

Le comportement de la fonction associée est décrit par le CBTRS suivant :

$$\begin{cases} reverse(nil, l) & \twoheadrightarrow \quad l \\ reverse(cons(x, l_1), l_2) & \twoheadrightarrow \quad reverse(l_1, cons(x, l_2)) \end{cases}$$

On peut également vérifier que l'inversion d'une liste inversement triée est triée (et donc qu'une inversion double ramène à une liste triée), en prenant l'annotation $\mathcal{P} = \{p_{sorted} * cons(B,l) \mapsto p_{inv}, cons(A,l) * p_{inv} \mapsto p_{inv}, p_{inv} * cons(A,l) \mapsto p_{sorted}, cons(B,l) * p_{sorted} \mapsto p_{sorted}, \}$.

C.5 Non-linéaire

On considère dans cet exemple les termes de l'algèbre définie par la signature $\Sigma_{nl} = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ décrite par les types algébriques :

$$\begin{array}{rcl} \mathsf{S1} & := & c(\mathsf{S2}, \mathsf{S2}) \\ & | & d_1(\mathsf{S2}, \mathsf{S2}) \\ & | & d_2(\mathsf{S1}) \end{array} \qquad\qquad \begin{array}{rcl} \mathsf{S2} & := & a \\ & | & b \end{array}$$

On étudie un système non-linéaire avec différentes règles dont la satisfaction de profil peut être vérifiée, ou non, en fonction de la méthode d'analyse choisie (par linéarisation, par évaluation stricte, ou via la sémantique substitutive).

On considère les symboles définis suivant :

$$\begin{array}{lcl} f^{!\mathcal{P}_f} & : & \mathsf{S1} * \mathsf{S2} \mapsto \mathsf{S1} \\ g & : & \mathsf{S1} \mapsto \mathsf{S2} \\ h^{!\mathcal{P}_h} & : & \mathsf{S2} * \mathsf{S2} \mapsto \mathsf{S1} \\ \varphi^{!\mathcal{P}_\varphi} & : & \mathsf{S2} * \mathsf{S2} \mapsto \mathsf{S2} \\ id^{!\mathcal{P}_{id}} & : & \mathsf{S2} \mapsto \mathsf{S2} \end{array}$$

avec $\mathcal{P}_f = \{\bot * \bot \mapsto c(a,b)\}$, $\mathcal{P}_h = \{b * \bot \mapsto c(a,b), \bot * a \mapsto c(a,b)\}$, $\mathcal{P}_\varphi = \{b * \bot \mapsto b, \bot * b \mapsto b\}$ et $\mathcal{P}_{id} = \{b \mapsto b\}$.

Le comportement de la fonction associée au symbole f est décrit par le CBTRS suivant :

$$\begin{cases} f(c(x,a),b) & \twoheadrightarrow \quad c(x,x) \\ f(c(x,b),a) & \twoheadrightarrow \quad h(x,x) \\ f(d(z),a) & \twoheadrightarrow \quad c(g(z),g(z)) \\ f(c(x,y),x) & \twoheadrightarrow \quad c(\varphi(x,y),\varphi(y,x))) \\ f(d(z),b) & \twoheadrightarrow \quad c(id(b),id(g(z))) \end{cases}$$

L'approche par linéarisation ne permet de prouver la préservation de sémantique d'aucune règle. L'approche par évaluation stricte permet de prouver que la première règle préserve la sémantique. L'approche par sémantique substitutive permet de prouver que les trois premières règles préservent la sémantique.

C.6 Réduction de motifs étendus

L'approche d'analyse statique permet également de vérifier la caractérisation des formes normales des motifs obtenues suivant l'approche proposée dans [CM19].

Pour cela, on considère une algèbre dont les termes représentent des motifs étendus avec les opérations de disjonction et de complément, et où les noms de variables et de symboles sont encodés par des entiers de Peano. La signature $\Sigma = (\mathcal{S}, \mathcal{C} \uplus \mathcal{D})$ de l'algèbre peut être décrite par les types algébriques suivant :

$$\begin{array}{rcl} \mathsf{Int} & := & z \\ & | & s(\mathsf{Int}) \\ \\ \mathsf{List} & := & \mathit{nil} \\ & | & \mathit{cons}(\mathsf{Expr}, \mathsf{List}) \end{array} \qquad \begin{array}{rcl} \mathsf{Expr} & := & \mathit{bot} \\ & | & \mathit{var}(\mathsf{Int}) \\ & | & \mathit{apply}(\mathsf{Int}, \mathsf{List}) \\ & | & \mathit{plus}(\mathsf{Expr}, \mathsf{Expr}) \end{array}$$

Le complément est encodé par un symbole défini annoté du profil $\bot * \bot \mapsto (\mathit{cons}(\mathit{bot} + \mathit{plus}(t_1, t_2), l) + \mathit{plus}(\mathit{bot}, t) + \mathit{plus}(t, \mathit{bot}))$ pour spécifier la caractérisation des formes normales données par la Proposition 2.3.

Le système considéré est un encodage du système $\mathfrak{R}_\setminus$ présenté en Figure 2.2.

Index

Bibliographie

[BBK+07] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom : Piggybacking rewriting on java. In *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007. doi:10.1007/978-3-540-73449-9_5. 2, 12, 30

[BCF03] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. Cduce : an XML-centric general-purpose language. In *Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, pages 51–63. ACM, 2003. doi:10.1145/944705.944711. 41, 162, 168

[BF99] Gilles Barthe and Maria João Frade. Constructor subtyping. In *Programming Languages and Systems, 8th European Symposium on Programming, ESOP '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS '99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1576 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 1999. doi:10.1007/3-540-49099-X_8. 40

[BGJR07] Yohan Boichut, Thomas Genet, Thomas P. Jensen, and Luka Le Roux. Rewriting approximations for fast prototyping of static analyzers. In *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2007. doi:10.1007/978-3-540-73449-9_6. 2, 12

[BKK+98] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In *1998 International Workshop on Rewriting Logic and its Applications, WRLA '98, Abbaye des Prémontrés at Pont-à-Mousson, France, September 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 55–70. Elsevier, 1998. doi:10.1016/S1571-0661(05)82552-6. 29

[BKM06] Emilie Balland, Claude Kirchner, and Pierre-Etienne Moreau. Formal islands. In *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Kuressaare, Estonia, July 5-8, 2006, Proceedings*, volume 4019 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2006. doi:10.1007/11784180_7. 30

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998. 15

[CELM96] Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of maude. In *First International Workshop on Rewriting Logic and its Applications,*

RWLW '96, Asilomar Conference Center, Pacific Grove, CA, USA, September 3-6, 1996, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89. Elsevier, 1996. doi:10.1016/S1571-0661(04)00034-9. 2, 12, 30

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000. doi:10.1007/10722167_15. 38

[CL20] Horatiu Cirstea and Pierre Lermusiaux. pfreecheck, 2020. [Online]. URL : https://github.com/plermusiaux/pfree_check. 14, 147, 191

[CLM15] Horatiu Cirstea, Sergueï Lenglet, and Pierre-Etienne Moreau. A faithful encoding of programmable strategies into term rewriting systems. In *26th International Conference on Rewriting Techniques and Applications, RTA 2015, Warsaw, Poland, June 29 - July 1, 2015*, volume 36 of *LIPIcs*, pages 74–88. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.RTA.2015.74. 8, 32, 41, 169

[CLM20a] Horatiu Cirstea, Pierre Lermusiaux, and Pierre-Etienne Moreau. Pattern eliminating transformations. In *Logic-Based Program Synthesis and Transformation, 30th International Symposium, LOPSTR 2020, Bologna, Italy, September 7–9, 2020, Proceedings*, volume 12561 of *Lecture Notes in Computer Science*, pages 74–92. Springer, 2020. doi:10.1007/978-3-030-68446-4_4. 47

[CLM20b] Horatiu Cirstea, Pierre Lermusiaux, and Pierre-Etienne Moreau. Preservation of Pattern-free properties for Constructor TRS. In *WPTE 2020 - 7th International Workshop on Rewriting Techniques for Program Transformations and Evaluation*, Paris, France, June 2020.

[CLM21] Horatiu Cirstea, Pierre Lermusiaux, and Pierre-Etienne Moreau. Static analysis of pattern-free properties. In *Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming, PPDP 2021, Tallinn, Estonia, September 6-8, 2021*, pages 9 :1–9 :13. ACM, 2021. doi:10.1145/3479394.3479404. 121, 157

[CM19] Horatiu Cirstea and Pierre-Etienne Moreau. Generic encodings of constructor rewriting systems. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 8 :1–8 :12. ACM, 2019. doi:10.1145/3354166.3354173. 3, 4, 17, 32, 33, 34, 43, 52, 57, 70, 72, 73, 85, 86, 87, 103, 165, 199

[FCB02] Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping. In *17th IEEE Symposium on Logic in Computer Science, LICS 2002, Copenhagen, Denmark, July 22-25, 2002, Proceedings*, pages 137–146. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029823. 8, 41, 167

[FGP+11] Carsten Fuhs, Jürgen Giesl, Michael Parting, Peter Schneider-Kamp, and Stephan Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2) :133–160, 2011. doi:10.1007/s10817-010-9215-9. 32

[FGT04] Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability analysis over term rewriting systems. *Journal of Automated Reasoning*, 33(3-4) :341–383, 2004. doi:10.1007/s10817-004-6246-0. 2, 12

[FP91] Timothy S. Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation, PLDI '91, Toronto, Ontario, Canada, June 26-28, 1991*, pages 268–277. ACM, 1991. doi:10.1145/113445.113468. 3, 40, 42, 162, 168

[Gar98] Jacques Garrigue. Programming with polymorphic variants. In *In ACM Workshop on ML*, 1998. 27

[GBB+17] Thomas Genet, Yohan Boichut, Benoît Boyer, Tristan Gillard, Timothée Haudebourg, and Sébastien Lê Cong. Reachability analysis and tree automata calculations, 2017. [Online]. URL : http://people.irisa.fr/Thomas.Genet/timbuk/. 159

[Gen14] Thomas Genet. Towards static analysis of functional programs using tree automata completion. In *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8663 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2014. doi:10.1007/978-3-319-12904-4_8. 2, 12, 27, 42, 168

[Gen16] Thomas Genet. Termination criteria for tree automata completion. *Journal of Logical and Algebraic Methods in Programming*, 85(1) :3–33, 2016. doi:10.1016/j.jlamp.2015.05.003. 37, 38, 159

[GR10] Thomas Genet and Vlad Rusu. Equational approximations for tree automata completion. *Journal of Symbolic Computation*, 45(5) :574–597, 2010. doi:10.1016/j.jsc.2010.01.009. 36, 37, 158

[Hau20] Timothée Haudebourg. Timbuk 4 : Regular verification framework based on tree automata and term rewriting system, 2020. [Online]. URL : https://gitlab.inria.fr/regular-pv/timbuk/timbuk. 158, 159

[HGJ20] Timothée Haudebourg, Thomas Genet, and Thomas P. Jensen. Regular language type inference with term rewriting. *Proceedings of the ACM on Programming Languages*, 4(ICFP) :112 :1–112 :29, 2020. doi:10.1145/3408994. 38, 162

[HKV12] Mark Hills, Paul Klint, and Jurgen J. Vinju. Program analysis scenarios in rascal. In *Rewriting Logic and Its Applications - 9th International Workshop, WRLA 2012, Held as a Satellite Event of ETAPS, Tallinn, Estonia, March 24-25, 2012, Revised Selected Papers*, volume 7571 of *Lecture Notes in Computer Science*, pages 10–30. Springer, 2012. doi:10.1007/978-3-642-34005-5_2. 31

[HP03] Haruo Hosoya and Benjamin C. Pierce. Xduce : A statically typed XML processing language. *ACM Transactions on Internet Technology*, 3(2) :117–148, 2003. doi:10.1145/767193.767195. 41

[JA07] Neil D. Jones and Nils Andersen. Flow analysis of lazy higher-order functional programs. *Theoretical Computer Science*, 375(1-3) :120–136, 2007. doi:10.1016/j.tcs.2006.12.030. 35

[JM79] Neil D. Jones and Steven S. Muchnick. Flow analysis and optimization of lisp-like structures. In *Proceeding of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '79, San Antonio, Texas, USA, January 29-31, 1979*, pages 244–256. ACM Press, 1979. doi:10.1145/567752.567776. 35

[Joh85] Thomas Johnsson. Lambda lifting : Treansforming programs to recursive equations. In *Functional Programming Languages and Computer Architecture, FPCA '85, Nancy, France, September 16-19, 1985, Proceedings*, volume 201 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 1985. doi:10.1007/3-540-15975-4_37. 41

[JR21] Eddie Jones and Steven J. Ramsay. Intensional datatype refinement : with application to scalable verification of pattern-match safety. *Proceedings of the ACM on Programming Languages*, 5(POPL) :1–29, 2021. doi:10.1145/3434336. 28

[KD13] Andrew W. Keep and R. Kent Dybvig. A nanopass framework for commercial compiler development. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming, ICFP 2013, Boston, MA, USA, September 25-27, 2013*, pages 343–350. ACM, 2013. doi:10.1145/2500365.2500618. 2, 12, 27

[Kee13] Andrew W. Keep. *A Nanopass Framework for Commercial Compiler Development*. PhD thesis, Indiana University, February 2013. 25

[KKK08] Claude Kirchner, Florent Kirchner, and Helene Kirchner. Strategic computation and deduction. In *Reasoning in Simple Type Theory. Festchrift in Honour of Peter B. Andrews on His 70th Birthday*, volume 17 of *Studies in Logic and the Foundations of Mathematics*, pages 339–364. College Publications, 2008. 20

[KKM07] Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-pattern matching. In *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4421 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2007. doi:10.1007/978-3-540-71316-6_9. 19, 34

[KKV93] Claude Kirchner, Hélène Kirchner, and Marian Vittek. Implementing computational systems with constraints. In *Principles and Practice of Constraint Programming, PPCP '93, Newport, Rhode Island, USA, April 28-30, 1993*, pages 156–165, 1993. 29

[Kra08] Alexander Krauss. Pattern minimization problems over recursive data types. In *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008, Victoria, BC, Canada, September 20-28, 2008*, pages 267–274. ACM, 2008. doi:10.1145/1411204.1411242. 33

[KRJ09] Ming Kawaguchi, Patrick Maxim Rondon, and Ranjit Jhala. Type-based data structure verification. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*, pages 304–315. ACM, 2009. doi:10.1145/1542476.1542510. 41

[KSU11] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 222–233. ACM, 2011. doi:10.1145/1993498.1993525. 39, 161

[KSZM09] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean termination tool 2. In *Rewriting Techniques and Applications, 20th International*

Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21. 32

[KTU10] Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 495–508. ACM, 2010. doi:10.1145/1706299.1706355. 39, 41

[KV10] Lennart C. L. Kats and Eelco Visser. The spoofax language workbench. In *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, Reno/Tahoe, Nevada, USA, October 17-21, 2010*, pages 237–238. ACM, 2010. doi:10.1145/1869542.1869592. 31

[KvdSV09] Paul Klint, Tijs van der Storm, and Jurgen J. Vinju. RASCAL : A domain specific language for source code analysis and manipulation. In *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, Alberta, Canada, September 20-21, 2009, Proceedings*, pages 168–177. IEEE Computer Society, 2009. doi:10.1109/SCAM.2009.28. 31

[LCM19] Pierre Lermusiaux, Horatiu Cirstea, and Pierre-Etienne Moreau. Pattern eliminating transformations. In *CIEL 2019 - 8ème Conférence en IngénieriE du Logiciel*, Toulouse, France, June 2019.

[MB04] José Meseguer and Christiano Braga. Modular rewriting semantics of programming languages. In *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004, Stirling, Scotland, UK, July 12-16, 2004, Proceedings*, volume 3116 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 2004. doi:10.1007/978-3-540-27815-3_29. 2, 12, 27

[MFP91] Erik Meijer, Maarten M. Fokkinga, and Ross Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Functional Programming Languages and Computer Architecture, 5th ACM Conference, FPCA '91, Cambridge, MA, USA, August 26-30, 1991, Proceedings*, volume 523 of *Lecture Notes in Computer Science*, pages 124–144. Springer, 1991. doi:10.1007/3540543961_7. 26

[Mil17] Milo Turner and Ohad Rau. Nanocaml, 2017. [Online]. URL : https://github.com/nanocaml/nanocaml. 2, 27

[MKU15] Yuma Matsumoto, Naoki Kobayashi, and Hiroshi Unno. Automata-based abstraction for automated verification of higher-order tree-processing programs. In *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2015. doi:10.1007/978-3-319-26529-2_16. 162

[MMV04] Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Towards a strategy language for maude. In *Proceedings of the Fifth International Workshop on Rewriting Logic and Its Applications, WRLA 2004, Barcelona, Spain, March 27-28, 2004*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier, 2004. doi:10.1016/j.entcs.2004.06.020. 30

[MSV03] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1) :66–97, 2003. doi:10.1016/S0022-0000(02)00030-2. 41

[Ong06] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21st IEEE Symposium on Logic in Computer Science, LICS 2006, Seattle, WA, USA, August 12-15, 2006, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38. 39, 42

[OR11] C.-H. Luke Ong and Steven J. Ramsay. Verifying higher-order functional programs with pattern-matching algebraic data types. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 587–598. ACM, 2011. doi:10.1145/1926385.1926453. 40, 161

[Rey68] John C. Reynolds. Automatic computation of data set definitions. In *Information Processing, Proceedings of IFIP Congress 1968, Edinburgh, UK, 5-10 August 1968, Volume 1 - Mathematics, Software*, pages 456–461, 1968. 168

[Rey93] John C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3-4) :233–248, 1993. doi:10.1007/BF01019459. 27

[RKJ08] Patrick Maxim Rondon, Ming Kawaguchi, and Ranjit Jhala. Liquid types. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2008, Tucson, AZ, USA, June 7-13, 2008*, pages 159–169. ACM, 2008. doi:10.1145/1375581.1375602. 3, 41, 42

[SF92] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming, LFP '92, San Francisco, California, USA, June 22-24, 1992*, pages 288–298. ACM, 1992. doi:10.1145/141471.141563. 27

[SKU+20] Ryosuke Sato, Naoki Kobayashi, Hiroshi Unno, Takuya Kuwahara, Keiichi Watanabe, and Naoki Iwayama. Mochi : Model checker for higher-order programs, 2020. [Online]. URL : http://www-kb.is.s.u-tokyo.ac.jp/~ryosuke/mochi/. 39, 161

[Spi01] Diomidis Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 56(1) :91–99, 2001. doi:10.1016/S0164-1212(00)00089-3. 30

[SWD04] Dipanwita Sarkar, Oscar Waddell, and R. Kent Dybvig. A nanopass infrastructure for compiler education. In *Proceedings of the ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, pages 201–212. ACM, 2004. doi:10.1145/1016850.1016878. 2, 27

[Ter03] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds. 15

[Toz06] Akihiko Tozawa. XML type checking using high-level tree transducer. In *Functional and Logic Programming, 8th International Symposium, FLOPS 2006, Fuji-Susono, Japan, April 24-26, 2006, Proceedings*, volume 3945 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2006. doi:10.1007/11737414_7. 41

[UTK10] Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Programming Languages*

and Systems, 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010, Proceedings, volume 6461 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2010. doi:10.1007/978-3-642-17164-2_22. 40, 161

[VBT98] Eelco Visser, Zine-El-Abidine Benaissa, and Andrew P. Tolmach. Building program optimizers with rewriting strategies. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming, ICFP '98, Baltimore, Maryland, USA, September 27-29, 1998*, pages 13–26. ACM, 1998. doi:10.1145/289423.289425. 29

[vdBvDH+01] Mark van den Brand, Arie van Deursen, Jan Heering, H. A. de Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter A. Olivier, Jeroen Scheerder, Jurgen J. Vinju, Eelco Visser, and Joost Visser. The ASF+SDF meta-environment : A component-based language development environment. In *Compiler Construction, 10th International Conference, CC 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2027 of *Lecture Notes in Computer Science*, pages 365–370. Springer, 2001. doi:10.1007/3-540-45306-7_26. 31

[Vis01] Eelco Visser. Stratego : A language for program transformation based on rewriting strategies. In *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–362. Springer, 2001. doi:10.1007/3-540-45127-7_27. 2, 12, 30

[VWT+14] Eelco Visser, Guido Wachsmuth, Andrew P. Tolmach, Pierre Neron, Vlad A. Vergu, Augusto Passalaqua, and Gabriël Konat. A language designer's workbench : A one-stop-shop for implementation and verification of language designs. In *Onward! 2014, Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, part of SPLASH 2014, Portland, OR, USA, October 20-24, 2014*, pages 95–111. ACM, 2014. doi:10.1145/2661136.2661149. 31

Résumé

La transformation de programmes est une pratique très courante dans le domaine des sciences informatiques. De la compilation à la génération de test en passant par de nombreuses approches d'analyse de codes et de vérification formelle des programmes, c'est un procédé qui est à la fois omniprésent et crucial au bon fonctionnement des programmes et systèmes informatiques. Cette thèse propose une étude formelle des procédures de transformation de programmes dans le but d'exprimer et de garantir des propriétés syntaxiques sur le comportement et les résultats d'une telle transformation.

Dans le contexte de la vérification formelle des programmes, il est en effet souvent nécessaire de pouvoir caractériser la forme des termes obtenus par réduction suivant une telle transformation. En s'inspirant du modèle de passes de compilation, qui décrivent un séquençage de la compilation d'un programme en étapes de transformation minimales n'affectant qu'un petit nombre des constructions du langage, on introduit, dans cette thèse, un formalisme basé sur les notions de filtrage par motif et de réécriture permettant de décrire certaines propriétés couramment induites par ce type de transformations.

Le formalisme proposé se repose sur un système d'annotations des symboles de fonction décrivant une spécification du comportement attendu des fonctions associées. On présente alors une méthode d'analyse statique permettant de vérifier que les transformations étudiées, exprimées par un système de réécriture, satisfont en effet ces spécifications.

Mots-clés: analyse statique, filtrage par motif, réécriture, sémantique de motif, spécification algébrique, transformation de programmes

Abstract

Program transformation is an extremely common practice in computer science. From compilation to tests generation, through many approaches of code analysis and formal verification of programs, it is a process that is both ubiquitous and critical to properly functioning programs and information systems. This thesis proposes to study the program transformations mechanisms in order to express and verify syntactical guarantees on the behaviour of these transformations and on their results.

Giving a characterisation of the shape of terms returned by such a transformation is, indeed, a common approach to the formal verification of programs. In order to express some properties often used by this type of approaches, we propose in this thesis a formalism inspired by the model of compilation passes, which are used to describe the general compilation of a program as a sequence of minimal transformations, and based on the notions of pattern matching and term rewriting.

This formalism relies on an annotation mechanism of function symbols in order to express a set of specifications describing the behaviours of the associated functions. We then propose a static analysis method in order to check that a transformation, expressed as a term rewrite system, actually verifies its specifications.

Keywords: algebraic specifications, pattern matching, pattern semantics, program transformation, static analysis, rewriting