Pattern eliminating transformations

Horatiu Cirstea, Pierre Lermusiaux, Pierre-Etienne Moreau

Pattern eliminating transformations

Horatiu Cirstea, Pierre Lermusiaux, and Pierre-Etienne Moreau
Université de Lorraine – LORIA
{name}.{surname}@loria.fr

Abstract

Program transformation is a common practice in computer science, and its many applications can have a range of different objectives. For example, a program written in an original high level language could be either translated into machine code for execution purposes, or towards a language suitable for formal verification. Such compilations are split into several so-called passes which generally aim at eliminating certain constructions of the original language to get some intermediate languages and finally generate the target code. Rewriting is a widely established formalism to describe the mechanism and the logic behind such transformations. In a typed context featuring type-preserving rewrite rules, the underlying type system can be used to give syntactic guarantees on the shape of the results obtained after each pass, but this approach could lead to an accumulation of (auxiliary) types that should be considered. We propose in this paper an approach where the function symbols corresponding to the transformations performed in a pass are annotated with the (anti-)patterns they are supposed to eliminate and show how we can check that the transformation is consistent with the annotations and thus, that it eliminates the respective patterns. With the generic principles governing term algebra and rewriting, we believe this approach to be an accurate formalism to any language providing pattern-matching primitives.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Rewriting, Pattern-matching, Pattern semantics, Compilation

1 Introduction

Rewriting is a well established formalism widely used in both computer science and mathematics. It has been used, for example, in semantics in order to describe the meaning of programming languages [25], but also in automated reasoning when describing, by inference rules, a logic, a theorem prover [19], or a constraint solver [18]. Rewriting has turned out to be particularly well adapted to describe program semantics [28] and program transformations [24, 7]. There are several languages and tools implementing the notions of pattern matching and rewriting rules ranging from functional languages, featuring relatively simple patterns and fixed rewriting strategies, to rule based languages like Maude [10], Stratego [31], or Tom [5], providing equational matching and flexible strategies; they have been all used as underlying languages for more or less sophisticated compilers.

In the context of compilation, the complete transformation is usually performed in multiple phases, also called passes, in order to eventually obtain a program in a different target language. Most of these passes concern transformations between some intermediate languages and often aim at eliminating certain constructions of the original language. These transformations could eliminate just some symbols, like in desugaring passes for example, or more elaborate construction, like in code optimization passes.

To guarantee the correctness of the transformations we could of course use runtime assertions but static guarantees are certainly preferable. When using typed languages, the types can be used to guarantee some of the constraints on the target language. In this case, the type of the function implicitly expresses the expected result of the transformation. The differences between the source and the target language concern generally only a small percentage of the symbols, and the definition of the target language is often tedious and contains a lot of the symbols from the source type. For example, for a pass performing

desugaring we would have to define a target language using the same symbols as the source one but the syntactic sugar symbols.

Formalisms such as the one proposed for NanoPass [20] have proposed a method to eliminate a lot of the overhead induced by the definition of the intermediate languages by specifying only the symbols eliminated from the source language and generating automatically the corresponding intermediate language.

For instance, let's consider expressions which are build out of (wrapped) integers, (wrapped) strings and lists:

$$\begin{array}{rcl} Expr & = & int(Int) \\ & | & str(String) \\ & | & lst(List) \end{array} \qquad \begin{array}{rcl} List & = & nil() \\ & | & cons(Expr, List) \end{array}$$

If, for some reason, we wand to define a pass encoding integers by strings then, the target language in NanoPass would be $Expr^{-int}$, *i.e.* expressions build out of strings and lists. Note that in this case the tool (automatically) removes the symbol *int* from $Expr$ and replaces accordingly $Expr$ with the new type in the type of *cons*.

This kind of approaches reach their limitations when the transformation of the source language go beyond the removal of some symbols. For example, if we want to define a transformation which flattens the list expressions and ensures thus that there is no nested list, the following target type should be considered:

$$\begin{array}{rcl} Expr & = & lit(Literal) \\ & | & lst(List) \end{array} \qquad \begin{array}{rcl} Literal & = & int(Int) \\ & | & str(String) \end{array} \qquad \begin{array}{rcl} List & = & nil() \\ & | & cons(Literal, List) \end{array}$$

Functional approaches to transformation [27] relying on the use of fine grained typing systems which combine overloading, subtyping and polymorphism through the use of variants [13] can be used to define the transformation and perform (implicitly) such verifications.

While effective, this method requires to design such adjusted types in a case by case basis. We propose in this paper a formalism where function symbols are annotated with the patterns that should be eliminated by the corresponding transformation and a mechanism to verify that the underlying rewriting system is consistent with the annotations. In our example, we could then keep the original types and just annotate the flattening transformation with the (anti-)pattern $cons(lst(l_1), l_2)$.

First, in the next section, we will introduce the basic notions and notations used in the article. We introduce then, in Section 3, the notion of pattern-free term and the corresponding ground semantics for general terms. Section 4 describes a method for checking pattern-freeness properties relying on the deep semantics, an extension of the ground semantics. In Section 5 we show how this method can be used to verify that specific patterns are absent from the result of a given transformation. We finally present some related work and conclude.

2 Preliminary notions

We define in this section the basic notions and notations used in this paper; more details can be found in [4, 30].

A *many-sorted signature* $\Sigma = (\mathcal{S}, \mathcal{F})$, consists of a set of sorts $\mathcal{S}$ and a set of symbols $\mathcal{F}$. We distinguish constructor symbols from function symbols by partitioning the alphabet $\mathcal{F}$ into $\mathcal{D}$, the set of *defined symbols*, and $\mathcal{C}$ the set of *constructors*. A symbol f with *domain* $\mathcal{D}om(f) = s_1 \times \ldots \times s_n \in \mathcal{S}^*$ and *co-domain* $\mathcal{C}o\mathcal{D}om(f) = s \in \mathcal{S}$ is written $f : s_1 \times \ldots \times s_n \mapsto s$; we may write f_s to indicate explicitly the co-domain. We denote by $\mathcal{C}_s$, resp. $\mathcal{D}_s$, the set of

constructors, resp. defined symbols, with co-domain s. Variables are also sorted and we write $x : s$ or x_s to indicate that variable x has sort s. The set $\mathcal{X}_s$ denotes a set of variables of sort s and $\mathcal{X} = \bigcup_{s \in \mathcal{S}} \mathcal{X}_s$ is the set of sorted variables.

The set of terms of sort $s \in \mathcal{S}$, denoted $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}_s$ and such that $f(t_1, \ldots, t_n)$ is in $\mathcal{T}_s(\mathcal{F}, \mathcal{X})$ whenever $f : s_1 \times \ldots \times s_n \mapsto s$ and $t_i \in \mathcal{T}_{s_i}(\mathcal{F}, \mathcal{X})$ for $i \in [1, n]$. We write $t : s$ to explicitly indicate that the term t is of sort s, *i.e.* when $t \in \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. The set of *sorted terms* is defined as $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \bigcup_{s \in \mathcal{S}} \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. The set of variables occurring in $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted by $\mathcal{V}ar\,(t)$. If $\mathcal{V}ar\,(t)$ is empty, t is called a *ground* term. $\mathcal{T}_s(\mathcal{F})$ denotes the set of all ground first-order terms of sort s and $\mathcal{T}(\mathcal{F})$ denotes the set of all ground first-order terms, while members of $\mathcal{T}(\mathcal{C})$ are called *values*. A *linear* term is a term where every variable occurs at most once. The linear terms in $\mathcal{T}(\mathcal{C}, \mathcal{X})$ are called *constructor patterns*.

A *position* of a term t is a finite sequence of positive integers describing the path from the root of t to the root of the subterm at that position. The empty sequence representing the root position is denoted by ε. $t_{|\omega}$, resp. $t(\omega)$, denotes the subterm of t, resp. the symbol of t, at position ω. $t\,[s]_\omega$ denotes the term t with the subterm at position ω replaced by s. We note $\mathcal{P}os\,(t)$ the set of positions of t.

We call *substitution* any mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ which is the identity except over a finite set of variables called its domain. A substitution σ extends as expected to an endomorphism σ' of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. To simplify the notations, we do not make the distinction between σ and σ'. Sorted substitutions are such that if $x : s$ then $\sigma(x) \in \mathcal{T}_s(\mathcal{F}, \mathcal{X})$. Note that for any such sorted substitution σ, $t : s$ iff $\sigma(t) : s$. In the context of a many-sorted algebra, we will only consider such sorted substitutions.

Given a sort s, a value $v : s$ and a constructor pattern p, we say that p *matches* v (denoted $p \lessdot v$) if there exists a substitution σ such that $v = \sigma(p)$. Since p is linear, we can also give an inductive definition to the pattern matching relation:

$$\begin{array}{rcll} x & \lessdot & v & x \in \mathcal{X} \\ c(p_1, \ldots, p_n) & \lessdot & c(v_1, \ldots, v_n) & \text{iff } \wedge_{i=1}^{n}\, p_i \lessdot v_i, \text{ for } c \in \mathcal{C} \end{array}$$

Starting from the observation that a pattern can be interpreted as the set of its instances, the notion of *ground semantics* was introduced in [9] as the set of all ground constructor instances of a pattern $p \in \mathcal{T}_s(\mathcal{C}, \mathcal{X})$: $[\![p]\!] = \{\sigma(p) \mid \sigma(p) \in \mathcal{T}_s(\mathcal{C})\}$. It can be shown [9] that, given a pattern p and a value v, $v \in [\![p]\!]$ iff $p \lessdot v$.

A *constructor rewrite rule* (over Σ) is a pair of terms $\varphi(l_1, \ldots, l_n) \twoheadrightarrow r \in \mathcal{T}_s(\mathcal{F}, \mathcal{X}) \times \mathcal{T}_s(\mathcal{F}, \mathcal{X})$ with $s \in \mathcal{S}$, $\varphi \in \mathcal{D}$, $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and such that $\varphi(l_1, \ldots, l_n)$ is linear and $\mathcal{V}ar\,(r) \subseteq \mathcal{V}ar\,(l)$. A *constructor term rewriting system* (CTRS) is a set of constructor rewrite rules $\mathcal{R}$ inducing a *rewriting relation* over $\mathcal{T}(\mathcal{F})$, denoted by $\longrightarrow_{\mathcal{R}}$ and such that $t \longrightarrow_{\mathcal{R}} t'$ iff there exist $l \twoheadrightarrow r \in \mathcal{R}$, $\omega \in \mathcal{P}os\,(t)$, and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$.

3 Pattern-free terms and corresponding semantics

As we have already said, we want to ensure that the normal form of a term, if it exists, does not contain a specific constructor and more generally that no subterm of this normal form matches a given pattern. The sort of the term provides some information on the shape of the normal form obtained when reducing it *w.r.t.* a sort preserving CTRS. Indeed, the precise language of the values of a given sort is implicitly given by the signature. Sometimes, it is possible to describe normal forms by providing guarantees stronger than the ones obtained

from the signature. Such guarantees thus depend not only on sorts but also on the underlying CTRS.

Since we want to check statically that some patterns never occur in a normal form we annotate all defined symbols with the patterns that are supposed to be absent when reducing a term headed by the respective symbol and we check that the CTRS defining the corresponding functions are consistent with these annotations.

We focus first on the notion of pattern-free term and on the corresponding ground semantics, and explain in the next sections how one can check pattern-free properties and subsequently verify the consistence of the symbol annotations with a given CTRS.

3.1 Pattern-free terms

We consider that every defined symbol $f \in \mathcal{D}$ is annotated with a pattern $p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X}) = \mathcal{T}(\mathcal{C}, \mathcal{X}) \cup \{\perp\}$ and we use this notation to define *pattern-free* terms. Intuitively, a ground term of the form $f^{-p}(t_1, \ldots, t_n)$ should ultimately be reduced to a value containing no subterms matched by p. Thus, if we consider the example given in the introduction, we can consider two function symbols, $flattenE^{-p} : Expr \mapsto Expr$ and $flattenL^{-p} : List \mapsto List$ with $p = cons(lst(l1), l2)$, to indicate that the normal forms of any term headed by one of these symbols contains no nested lists. On the other hand, the annotation of the function symbol for the concatenation, $concat^{-\perp} : List \times List \mapsto List$, indicates that no particular shape is expected for the reducts of the corresponding terms.

▶ **Definition 3.1** (Pattern-free terms). *Given a pattern $p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X})$,*

- *a value $v \in \mathcal{T}(\mathcal{C})$ is p-free iff $\forall \omega \in \mathcal{P}os(v), p \not\trianglelefteq v_{|\omega}$;*
- *a term $t[f_s^{-q}(t_1, \ldots, t_n)]_\omega \in \mathcal{T}(\mathcal{F})$ is p-free iff $\forall v \in \mathcal{T}_s(\mathcal{C})$ q-free, $t[v]_\omega$ is p-free;*
- *a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is p-free iff $\forall \sigma$ such that $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $\sigma(t)$ is p-free.*

A value is p-free if and only if p matches no subterm of the value. A ground term is p-free if and only if replacing (all) the subterms headed by a defined symbol f_s^{-q} by any q-free value of the same sort s results in a p-free term. Intuitively, this corresponds to considering an over-approximation of the set of normal forms of an annotated term. For general terms, verifying a pattern-free property comes to verifying the property for all the ground instances of the term. While, in the case of a value, it can be checked by exploring all its subterms, this is not possible for a general term since the property have to be verified by a potentially infinite number of values. We present in Section 4.2 an approach for solving this problem.

We can already note that in some cases pattern-free properties can actually be implied by the sort and when all terms of sort s are p-free we say that s *excludes* p. We can indeed establish such a property by observing that s excludes any pattern whose sort is not in the language of s. Moreover, given two constructor patterns p and q, we say that p *destroys* q iff any p-free term is also q-free.

▶ **Example 3.2.** Consider the signature Σ with $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{F} = \mathcal{C} = \{c_1 : s_2 \times s_1 \mapsto s_1, c_2 : s_3 \mapsto s_1, c_3 : s_1 \mapsto s_2, c_4 : s_3 \mapsto s_2, c_5 : s_3 \mapsto s_3, c_6 : [] \mapsto s_3\}$.

It is easy to see that the only reachable sort from s_3 is s_3, *i.e.* any value of sort s_3 can only have subterms of sort s_3. Therefore, s_3 excludes all the patterns of the other sorts. All the sorts are reachable from s_1 and s_2 and thus, these sorts exclude no pattern.

One can also observe that $c_4(z)$ destroys $c_1(c_4(z), y)$, since any term that does not contain any instance of $c_4(z)$ cannot contain an instance of $c_1(c_4(z), y)$. Reciprocally, $c_1(c_4(z), y)$ does not destroy $c_4(z)$ as we can easily construct a term that is $c_1(c_4(z), y)$-free but not $c_4(z)$-free (like $c_4(c_6())$ for example).

3.2 Generalized ground semantics

The notion of ground semantics presented in Section 2 and, in particular, the approach proposed in [9] to compute differences (and thus intersections) of such semantics, can be used to compare the shape of two constructor patterns p, q (at the root position). More precisely, when $[\![p]\!] \cap [\![q]\!] = \emptyset$ we have that $\forall \sigma, \sigma(q) \notin [\![p]\!]$ and therefore, we can establish that $\forall \sigma, p \not\trianglelefteq \sigma(q)$. We can thus compare the semantics of a given pattern p with the semantics of each of the subterms of a term t in order to check that t is p-free.

▶ **Example 3.3.** We consider the signature from Example 3.2. We can remark that $[\![c_3(c_2)]\!] \subseteq [\![c_3(x)]\!]$ and thus that $c_3(c_2)$ is redundant *w.r.t.* $c_3(x)$; consequently $c_3(c_2)$ is not $c_3(x)$-free. Moreover, we have $[\![c_1(c_4(c_6), y)]\!] \cap [\![c_1(x, c_2(c_6))]\!] = [\![c_1(c_4(c_6), c_2(c_6))]\!]$ and thus neither $c_1(c_4(c_6), y)$ is $c_1(x, c_2(c_6))$-free nor $c_1(x, c_2(c_6))$ is $c_1(c_4(c_6), x)$-free.

Similarly, we can check that $[\![c_3(c_2)]\!] \cap [\![c_4(z)]\!] = \emptyset$ and $[\![c_2]\!] \cap [\![c_4(z)]\!] = \emptyset$ and consequently, we can deduce that $c_3(c_2)$ is $c_4(z)$-free.

The pattern-free properties in the above example could have been checked through pattern matching by checking the subsumption relations between all the subterms and the considered pattern. However, we actually want to establish a general method to verify pattern-free properties for any term and we propose an approach which largely relies on the notion of ground semantics introduced in [9] extended to take into account all terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$:

▶ **Definition 3.4** (Generalized ground semantics)**.** *Given a sort $s \in \mathcal{S}$ and a pattern $p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X})$*

- $[\![f_s^{-p}(t_1, \ldots, t_n)]\!] = \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \wedge v \text{ } p\text{-free}\}, \forall f_s^{-p} \in \mathcal{D}_s$
- $[\![c(t_1, \ldots, t_n)]\!] = \{c(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in [\![t_1]\!] \times \ldots \times [\![t_n]\!]\}, \forall c \in \mathcal{C}$
- $[\![x_s]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(x_1, \ldots, x_n)]\!]$.

The generalized ground semantics of a term rooted by a defined symbol represents an over-approximation of all the possible values obtained by reducing the term with respect to a TRS preserving the pattern-free properties, by taking into account the annotation of the respective defined symbol. Note that this is a proper generalization of the definition of the ground semantics in [9] and the two definitions are equivalent when restricted to constructor terms.

For convenience, we consider also annotated variables whose semantics is that of any term headed by a defined symbol with the same co-domain as the sort of the variable:

$$[\![x_s^{-p}]\!] = \{v \mid v \in \mathcal{T}_s(\mathcal{C}) \wedge v \text{ } p\text{-free}\}$$

Thus, $[\![f_s^{-p}(t_1, \ldots, t_n)]\!] = [\![x_s^{-p}]\!], \forall f_s^{-p} \in \mathcal{D}_s$. Note that $x_s^{-\perp}$ has the same semantics as x_s. We denote by $\mathcal{X}^a$ the set of annotated variables.

Moreover, given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we can systematically construct its *symbolic equivalent* $\tilde{t} \in \mathcal{T}(\mathcal{C}, \mathcal{X}^a)$ by replacing all the subterms of t headed by a defined symbol by a fresh variable of the corresponding sort and similarly annotated:

▶ **Proposition 3.5.** $\forall t \in \mathcal{T}(\mathcal{F}, \mathcal{X}), [\![t]\!] = [\![\tilde{t}]\!]$

We can thus restrict in what follows to patterns using annotated variables and, as in [9], we consider *extended patterns* built out of this kind of patterns:

$$p \quad := \quad \mathcal{X}^a \mid c(q_1, \ldots, q_n) \mid p_1 + p_2 \mid p_1 \setminus p_2 \mid p_1 \times p_2 \mid \perp$$

with $p, p_1, p_2 : s$ for some $s \in \mathcal{S}, c : s_1 \times \ldots \times s_n \mapsto s \in \mathcal{C}$ and $\forall i \in [1, n], q_i : s_i$

The pattern matching relation can be extended to take into account disjunctions, conjunctions and complements of patterns:

$$\begin{array}{rcll} p_1 + p_2 & \lessdot & v & \text{iff } p_1 \lessdot v \ \vee \ p_2 \lessdot v \\ p_1 \setminus p_2 & \lessdot & v & \text{iff } p_1 \lessdot v \ \wedge \ p_2 \not\lessdot v \end{array} \qquad \begin{array}{rcll} p_1 \times p_2 & \lessdot & v & \text{iff } p_1 \lessdot v \ \wedge \ p_2 \lessdot v \\ \bot & \not\lessdot & v & \end{array}$$

Intuitively, a pattern $p_1 + p_2$ matches any term matched by one of its components while a pattern $p_1 \times p_2$ matches any term matched by both its components. The relative complement of p_2 *w.r.t.* p_1, $p_1 \setminus p_2$, matches all terms matched by p_1 but those matched by p_2. $\bot$ matches no term. $\times$ has a higher priority than $\setminus$ which has a higher priority than $+$.

The notion of ground semantics can be also adapted to handle such patterns:

$$[\![p_1 + p_2]\!] = [\![p_1]\!] \cup [\![p_2]\!] \qquad [\![p_1 \setminus p_2]\!] = [\![p_1]\!] \setminus [\![p_2]\!] \qquad [\![p_1 \times p_2]\!] = [\![p_1]\!] \cap [\![p_2]\!] \qquad [\![\bot]\!] = \emptyset$$

In this context, if an extended pattern contains no $\bot$ it is called *pure*, if it contains no $\times$ and no $\setminus$ it is called *additive*, and if it contains no $+$, no $\times$ and no $\setminus$, *i.e.* a term of $\mathcal{T}(\mathcal{C}, \mathcal{X}^a)$, it is called *symbolic*. We call *regular* patterns that contain only variables of the form $x^{-\bot}$. And finally, we call *quasi-additive* patterns that contain no $\times$ and only contains $\setminus$ with the pattern on the left being a variable and the pattern on the right being a regular additive pattern.

▶ **Example 3.6.** We consider the signature from Example 3.2 enriched with the defined symbols $\mathcal{D} = \{f : s_1 \mapsto s_1, g : s_2 \mapsto s_2\}$ such that f is supposed to eliminate the pattern $p_1 = c_1(c_4(z), y)$ and g the pattern $p_2 = c_4(z)$ (the corresponding TRS will be presented in Section 5).

If we consider consider the term $r_1 = c_1(g^{-p_2}(x), f^{-p_1}(y))$, to construct its symbolic equivalent, we replace $f^{-p_1}(y)$ and $g^{-p_2}(x)$ by $y_{s_1}^{-p_1}$ and $x_{s_2}^{-p_2}$, respectively. Thus we have $\tilde{r_1} = c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})$. Similarly, for $r_2 = c_3(f(y))$, we have $\tilde{r_2} = c_3(y_{s_1}^{-p_1})$.

We can also remark that p_1 and p_2 are regular patterns, that $c_1(x_{s_2}^{-p_2} \setminus p_2, y_{s_1}^{-p_1})$ is a quasi-additive pattern, and that, as for all symbolic equivalents, $\tilde{r_1}$ and $\tilde{r_2}$ are symbolic patterns.

▶ **Proposition 3.7.** *Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X}), p \in \mathcal{T}_\bot(\mathcal{C}, \mathcal{X})$, t is p-free iff $\forall v \in [\![t]\!], v$ is p-free.*

While this gives a more straightforward condition to the pattern-free property, by simply having to check if every member of the ground semantics respects the pattern-free property, there is still a possible infinite number of values to check.

That being said, the ground semantics was explicitly introduced in [9] as a means to represent an infinite number of terms by a finite structure and to subsequently solve similar problems. As stated in Proposition 3.7 checking that a term t is p-free comes to verify that each value v in its ground semantics is p-free or equivalently that $\forall v \in [\![t]\!], \forall \omega \in \mathcal{P}os(v), p \not\lessdot v_{|\omega}$, *i.e.* that each value and each of its sub-terms is not matched by the pattern p. However, the notion of ground semantics only provides a finite structure capable of representing the potentially infinite set of instances of a term while here would need an extended notion of ground semantics closed by the sub-term relation.

4 Deep semantics for pattern-free properties

We introduce now an extended notion of ground semantics satisfying the above requirements, show how it can be expressed in terms of ground semantics, and eventually provide a method for checking the emptiness of the intersection of such semantics and thus, assert pattern-free properties.

4.1 Deep semantics

The ground semantics of a term provides information essentially on the shape of the term at the root position, and, as such, is not adequately suited to analyse pattern-free properties that constrain the shape at all positions of the term. We introduce thus, the notion of *deep semantics* which provides more comprehensive information on the shape of the (sub-)terms.

▶ **Definition 4.1.** *(Deep semantics) Let t be an extended pattern, its deep semantics $\{\!\{t\}\!\}$ is defined as follows:*

$$\{\!\{t\}\!\} = \{u_{|\omega} \mid u \in [\![t]\!], \omega \in \mathcal{P}os\,(u)\}$$

Note first that, similarly to the case of generalized ground semantics, it is obvious that we can always exhibit a symbolic pattern equivalent in terms of deep semantics to a given term, *i.e.* $\forall t \in \mathcal{T}(\mathcal{F}, \mathcal{X}), \{\!\{t\}\!\} = \{\!\{\widetilde{t}\}\!\}$; consequently, we can focus on the computation of the deep semantics of extended patterns. Following this observation and as an immediate consequence of the definition we have a necessary and sufficient condition with regards to pattern-free properties:

▶ **Proposition 4.2** (Pattern-free vs Deep Semantics)**.** *Let $p \in \mathcal{T}(\mathcal{C}, \mathcal{X}), t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, t is p-free iff $\{\!\{\widetilde{t}\}\!\} \cap [\![p]\!] = \emptyset$.*

To check that the intersection in Proposition 4.2 is empty we first express, as shown below, the deep semantics of a term as a union of ground semantics that we can then compare one by one to the semantics of the considered pattern to verify emptiness.

Since the deep semantics is based on the generalized ground semantics, we can easily establish a similar recursive definition for constructor patterns:

▶ **Proposition 4.3.** *For any pattern $p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X})$, constructor symbol $c \in \mathcal{C}$ such that $c : s_1 \times \ldots \times s_n \mapsto s$, and terms $(t_1, \ldots, t_n) \in \mathcal{T}_{s_1}(\mathcal{C}, \mathcal{X}^a) \times \ldots \times \mathcal{T}_{s_n}(\mathcal{C}, \mathcal{X}^a)$, we have:*

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

For annotated variables we have $\{\!\{x_s^{-p}\}\!\} = \{u_{|\omega} \mid u \in [\![x_s^{-p}]\!], \omega \in \mathcal{P}os\,(u)\}$ and we can first observe that the ground semantics of an annotated variable can be also defined as:

$$[\![x_s^{-p}]\!] = \bigcup_{c \in \mathcal{C}_s} [\![c(x_{s_1}^{-p}, \ldots, x_{s_i}^{-p}) \setminus p]\!] \tag{1}$$

By distributing the complement pattern on the subterms (see rule $M7$ in Figure 2), each of the elements of the above union can be expressed as

$$[\![c(x_{s_1}^{-p}, \ldots, x_{s_n}^{-p}) \setminus p]\!] = \Big[\!\Big[\sum_{q \in Q_c(p)} c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n) \Big]\!\Big] = \bigcup_{q \in Q_c(p)} [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!]$$

with $Q_c(p)$ a set of tuples $q = (q_1, \ldots, q_n)$ of patterns, with each q_i being either $\perp$ or a subterm of p. For each $q = (q_1, \ldots, q_n)$ such that $[\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!] \neq \emptyset$ we have

$$\{\!\{c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)\}\!\} = [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!] \cup \bigcup_{i=1}^{n} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}$$

We denote $Q'_c(p)$ the set of all the tuples q satisfying the non-emptiness property and we obtain

$$\begin{array}{rcl}
\{\!\{x_s^{-p}\}\!\} & = & \{u_{|\omega} \mid u \in [\![x_s^{-p}]\!], \omega \in \mathcal{P}os(u)\} \\
& = & \left\{ u_{|\omega} \mid u \in \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!], \omega \in \mathcal{P}os(u) \right\} \\
& = & \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} \{u_{|\omega} \mid u \in [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!], \omega \in \mathcal{P}os(u)\} \\
& = & \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} \{\!\{c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)\}\!\} \qquad (2) \\
& = & \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} [\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!] \quad \cup \quad \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} \bigcup_{i=1}^{n} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\} \\
& = & [\![x_s^{-p}]\!] \quad \cup \quad \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(p)} \bigcup_{i=1}^{n} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}
\end{array}$$

We can remark from this development that the $x_{s_i}^{-p} \setminus q_i$ terms obtained represent the possible direct subterms of any term matched by the variable. Hence the deep semantics of a variable is the union of its ground semantics with the deep semantics of these terms.

▶ **Example 4.4.** We consider the symbolic patterns from Example 3.6 and express their deep semantics as a union of the form explained above. We have $\{\!\{c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})\}\!\} = [\![c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})]\!] \cup \{\!\{x_{s_2}^{-p_2}\}\!\} \cup \{\!\{y_{s_1}^{-p_1}\}\!\}$. Thus we now want to expand $\{\!\{x_{s_2}^{-p_2}\}\!\}$ and $\{\!\{y_{s_1}^{-p_1}\}\!\}$.

In order to expand $\{\!\{y_{s_1}^{-p_1}\}\!\}$, we have to compute, $\forall c \in \mathcal{C}_s$, the corresponding sets $Q_c(p_1)$ as shown in the development (2). We have $\mathcal{C}_{s_1} = \{c_1, c_2\}$ and thus, according to (1)

$$[\![y_{s_1}^{-p_1}]\!] = [\![c_1(x_{s_2}^{-p_1}, y_{s_1}^{-p_1}) \setminus c_1(c_4(z_{s_3}^{-\perp}), y_{s_1}^{-\perp})]\!] \cup [\![c_2(z_{s_3}^{-p_1}) \setminus c_1(c_4(z_{s_3}^{-\perp}), y_{s_1}^{-\perp})]\!]$$

We can easily show that the complement relation in term of ground semantics corresponds to set differences of cartesian products: $[\![c_1(x_{s_2}^{-p_1}, y_{s_1}^{-p_1}) \setminus c_1(c_4(z_{s_3}^{-\perp}), y_{s_1}^{-\perp})]\!] = [\![c_1(x_{s_2}^{-p_1} \setminus c_4(z_{s_3}^{-\perp}), y_{s_1}^{-p_1})]\!] \cup [\![c_1(x_{s_2}^{-p_1}, y_{s_1}^{-p_1} \setminus y_{s_1}^{-\perp})]\!]$ and, as $p_2 = c_4(z_{s_3}^{-\perp})$, we then get:

$$[\![y_{s_1}^{-p_1}]\!] = [\![c_1(x_{s_2}^{-p_1} \setminus p_2, y_{s_1}^{-p_1})]\!] \cup [\![c_1(x_{s_2}^{-p_1}, y_{s_1}^{-p_1} \setminus y_{s_1}^{-\perp})]\!] \cup [\![c_2(z_{s_3}^{-p_1})]\!]$$

Hence $Q_{c_1}(p_1) = \{(p_2, \perp), (\perp, y_{s_1})\}$ and $Q_{c_2}(p_1) = \{(\perp)\}$. Moreover, $[\![c_1(x_{s_2}^{-p_1} \setminus p_2, y_{s_1}^{-p_1})]\!]$ and $[\![c_2(z_{s_3}^{-p_1})]\!]$ are not empty ($c_1(c_3(c_2(c_6)), c_2(c_6))$ and $c_2(c_6)$ belong respectively to each of them) while $[\![c_1(x_{s_2}^{-p_1}, y_{s_1}^{-p_1} \setminus y_{s_1}^{-\perp})]\!]$ is clearly empty. Thus, as shown in (2), we have

$$\{\!\{y_{s_1}^{-p_1}\}\!\} = [\![y_{s_1}^{-p_1}]\!] \cup \{\!\{x_{s_2}^{-p_1} \setminus p_2\}\!\} \cup \{\!\{y_{s_1}^{-p_1}\}\!\} \cup \{\!\{z_{s_3}^{-p_1}\}\!\}.$$

Similarly, for $\{\!\{x_{s_2}^{-p_2}\}\!\}$ we have $[\![x_{s_2}^{-p_2}]\!] = [\![c_3(y_{s_1}^{-p_2}) \setminus p_2]\!] \cup [\![c_4(z_{s_3}^{-p_1}) \setminus p_2]\!] = [\![c_3(y_{s_1}^{-p_2})]\!] \cup [\![c_4(z_{s_3}^{-p_1} \setminus z_{s_3}^{-\perp})]\!]$ and hence $Q_{c_3}(p_2) = \{(\perp)\}$ and $Q_{c_4}(p_2) = \{(z_{s_3})\}$. Moreover, $[\![c_3(y_{s_1}^{-p_2})]\!]$ is not empty ($c_3(c_2(c_6))$ belongs to it) while $[\![c_4(z_{s_3}^{-p_1} \setminus z_{s_3}^{-\perp})]\!]$ is clearly empty. Thus we have

$$\{\!\{x_{s_2}^{-p_2}\}\!\} = [\![x_{s_2}^{-p_2}]\!] \cup \{\!\{y_{s_1}^{-p_2}\}\!\}.$$

We can now remark that, by considering x_s^{-p} as $x_s^{-p} \setminus \perp$, we can generalise the development (2) to calculate the deep semantics of a pattern of the form $x_s^{-p} \setminus r$ where r is either $\perp$ or a sum of constructor patterns. In order to express the deep semantics of annotated variables as a union of ground semantics we thus have to compute a fixpoint for the equation

$$\{\!\{x_s^{-p} \setminus r\}\!\} = [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(r+p)} \bigcup_{i=1}^{n} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}$$

Function `getReachable`(s, p, S, r)

Data: s: current sort,
p: pattern of the pattern-free property,
S: set of couples (s', p') reached,
r: resulting pattern

Result: set of couples (s', p') reachable from $x_s^{-p} \setminus r$

if $p : s$ **then** $r \longleftarrow r + p$
if $[\![x_s \setminus r]\!] = \emptyset$ **then return** $\emptyset$
if $\exists\,(s, r') \in S, [\![r']\!] = [\![r]\!]$ **then return** S
$R \longleftarrow S \cup \{(s, r)\}$
$reachable \longleftarrow False$
for $c \in \mathcal{C}_s$ **do**
  $Q_c \longleftarrow \{(\overbrace{\bot, \ldots, \bot}^{m})\}$ with $m = arity(c)$
  for $i = 1$ **to** n *with* $r = \sum_{i=1}^{n} r_i$ **do**
    if $r_i(\omega) = c$ **then**
      $tQ_c \longleftarrow \emptyset$
      for $(q_1, \ldots, q_m) \in Q_c, k \in [1, m]$ **do** $tQ_c.add((q_1, \ldots, q_k + r_{i|k}, \ldots, q_m))$
      $Q_c \longleftarrow tQ_c$
  for $(q_1, \ldots, q_m) \in Q_c$ **do**
    $subRset \longleftarrow \emptyset$
    for $i = 1$ **to** m **do**
      $subR \longleftarrow$ `getReachable`$(\mathcal{D}om\,(c)\,[i], p, R, q_i)$
      if $subR \neq \emptyset$ **then** $subRset.add(subR)$
    if $|subRset| = m$ **then**
      $reachable \longleftarrow True$
      for $subR \in subRset$ **do** $R \longleftarrow R \cup subR$
if *reachable* **then**
  return R
else
  return $\emptyset$

Figure 1 Compute deep semantics of quasi-additive terms as a union of ground semantics.

Given a sort s, a constructor pattern p and a sum of constructor patterns r, we propose a method to determine the set R of pairs (s', p'), with $s' \in \mathcal{S}$ and p' being either $\bot$ or a sum of constructor patterns, *reachable* from $x_s^{-p} \setminus r$, *i.e.* such that we have:

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

More precisely, we propose the algorithm `getReachable` presented in Figure 1 which computes this set. Note that when r is $\bot$ the algorithm computes the set of tuples reachable from x_s^{-p}.

The general idea of the algorithm is to compute a fixpoint of the reachable pairs (s', r') from $x_s^{-p} \setminus r$. This is achieved through a recursion on the pairs (s', r') obtained by distributing r and p (if $p : s$) on the constructors $c \in \mathcal{C}_s$ as done in (2) and illustrated in Example 4.4. To ensure the termination of the algorithm we use the set S to store the pairs that have been reached through the algorithm, and as any component of a tuple from Q_c is either $\bot$, or a sum of subterms of p or r, the r' obtained this way can therefore only be $\bot$ or a sum of

subterms of p or of the original r (of which there is only a finite possible number).

▶ **Proposition 4.5** (Correctness). *Let $s \in \mathcal{S}, p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X})$ and r : s a sum of constructor patterns, if $R = \mathit{getReachable}(s, p, \emptyset, r)$, then*

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

Moreover, we have $\{\!\{x_s^{-p} \setminus r\}\!\} = \emptyset$ iff $R = \emptyset$.

▶ **Example 4.6.** The developements presented in Example 4.4 are implemented by the `getReachable` algorithm which eventually computes the fix-point of these equations, allowing thus to express the deep semantics of any term of the form $x_s^{-p} \setminus r$ as a union of ground semantics. More precisely, we get $\{\!\{y_{s_1}^{-p_1}\}\!\} = [\![y_{s_1}^{-p_1}]\!] \cup [\![z_{s_3}^{-p_1}]\!] \cup [\![x_{s_2}^{-p_1} \setminus p_2]\!]$ and $\{\!\{x_{s_2}^{-p_2}\}\!\} = [\![x_{s_2}^{-p_2}]\!] \cup [\![y_{s_1}^{-p_2}]\!] \cup [\![z_{s_3}^{-p_2}]\!]$, and therefore, the deep semantics of $\tilde{r_1} = c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})$ is the union of $[\![c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})]\!]$, $[\![y_{s_1}^{-p_1}]\!]$, $[\![z_{s_3}^{-p_1}]\!]$, $[\![x_{s_2}^{-p_1} \setminus p_2]\!]$, $[\![x_{s_2}^{-p_2}]\!]$, $[\![y_{s_1}^{-p_2}]\!]$ and $[\![z_{s_3}^{-p_2}]\!]$. Each of these ground semantics can be compared to p_1 to check that r_1 is p_1-free:

- $[\![y_{s_1}^{-p_1}]\!] \cap [\![p_1]\!]$, $[\![z_{s_3}^{-p_1}]\!] \cap [\![p_1]\!]$ and $[\![(x_{s_2}^{-p_1} \setminus p_2)]\!] \cap [\![p_1]\!]$ are all empty by definition of the semantics of x^{-p_1}
- Similarly, $[\![x_{s_2}^{-p_2}]\!] \cap [\![p_1]\!]$, $[\![y_{s_1}^{-p_2}]\!] \cap [\![p_1]\!]$ and $[\![z_{s_3}^{-p_2}]\!] \cap [\![p_1]\!]$ are empty, because p_2 destroys p_1.
- Finally, as $p_1 = c_1(p_2, y_{s_1})$, we have $[\![c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})]\!] \cap [\![p_1]\!] = \{c_1(t_1, t_2) | t_1 \in [\![x_{s_2}^{-p_2}]\!] \cap [\![p_2]\!], t_2 \in [\![y_{s_1}^{-p_1}]\!] \cap [\![y_{s_1}^{-\perp}]\!]\}$, and as $[\![x_{s_2}^{-p_2}]\!] \cap [\![p_2]\!]$ is empty, so is $[\![c_1(x_{s_2}^{-p_2}, y_{s_1}^{-p_1})]\!] \cap [\![p_1]\!]$.

Thus, the Propositions 4.3 and 4.5 guarantee that the deep semantics of any symbolic pattern and thus, of any term, can actually be expressed as the union of ground semantics of quasi-additive patterns. As we have seen in Example 4.6, it is then possible to check that the corresponding intersections with the semantics of a given pattern p are empty in order to prove that a term is p-free. We will propose in the next section a method to automatically verify that such intersections are indeed empty.

4.2 Establishing pattern-free properties

As we have already mentioned, an approach was proposed in [9] to compute the intersection of ground semantics for constructor patterns. More precisely, a convergent TRS was used to reduce any extended pattern (without annotations) into an equivalent sum, potentially empty, of constructor patterns. We should point out that the conjunction in [9] was restricted to patterns of the form $x \times p$ with x a (not annotated) variable.

A TRS naively adapted from the one in [9] to take into account annotated variables would not be terminating, as explained later on. Moreover, compared to [9], in our case we just need to check that the intersection of the semantics of quasi-additive patterns with the semantics of a given pattern p is empty: to put it simply, we want a TRS that reduces a pattern of the form $t \times p$, with t a quasi-additive pattern and p a constructor pattern, to $\perp$ if and only if its ground semantics is empty.

To this end, we introde the TRS $\mathcal{R}_p$ presented in Figure 2. Most of the rules are inherited from the system introduced in [9], the new ones concerning the conjunction and relations with annotated variables. The rules generally correspond to their counterparts from set theory where constructor patterns correspond to cartesian products and the other extended patterns to the obvious set operations.

The rules $A1$, $A2$, resp. $E2$, $E3$, describe the behaviour of the conjunction, resp. the disjunction, *w.r.t.* $\perp$. Rule $E1$ indicates that the semantics of a pattern containing a sub-term

Remove empty sets:

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

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

Distribute sets:

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

(E2) $\bot \times \overline{v} \Rightarrow \bot$

(E3) $\overline{v} \times \bot \Rightarrow \bot$

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

(S3) $\overline{w} \times (\overline{v_1} + \overline{v_2}) \Rightarrow (\overline{w} \times \overline{v_1}) + (\overline{w} \times \overline{v_2})$

Simplify complements:

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

Simplify conjunctions:

(T1) $\overline{v} \times \overline{x_s}^{-\bot} \Rightarrow \overline{v}$

(T2) $\overline{x_s}^{-\bot} \times \overline{v} \Rightarrow \overline{v}$

(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) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \beta(\overline{w_1}, \ldots, \overline{w_m}) \Rightarrow \bot$ *with* $\alpha \neq \beta$

Simplify p-free:

(P1) $\overline{x_s}^{\overline{p}} \times \alpha(\overline{v_1}, \ldots, \overline{v_n}) \Rightarrow \sum_{c \in \mathcal{C}_s} c(z_1{}_{s_1}^{-p}, \ldots, z_m{}_{s_m}^{-p}) \times (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \setminus \overline{p})$ *with* $m = arity(c)$

(P2) $\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times (\overline{x_s}^{\overline{p}} \setminus \overline{t}) \Rightarrow (\alpha(\overline{v_1}, \ldots, \overline{v_n}) \times \overline{x_s}^{\overline{p}}) \setminus \overline{t}$ if $\{\!|\overline{x_s}^{\overline{p}} \setminus \overline{t}|\!\} \neq \emptyset$

(P3) $\overline{x_s}^{\overline{q}} \times (\overline{x_s}^{\overline{p}} \setminus \overline{t}) \Rightarrow (\overline{x_s}^{\overline{q}} \times \overline{x_s}^{\overline{p}}) \setminus \overline{t}$ if $\{\!|\overline{x_s}^{\overline{p}} \setminus \overline{t}|\!\} \neq \emptyset$

(P4) $(\overline{x_s}^{\overline{p}} \setminus \overline{t}) \times \overline{v} \Rightarrow (\overline{x_s}^{\overline{p}} \times \overline{v}) \setminus \overline{t}$ if $\{\!|\overline{x_s}^{\overline{p}} \setminus \overline{t}|\!\} \neq \emptyset$

(P5) $(\overline{x_s}^{\overline{p}} \setminus \overline{t}) \setminus \overline{u} \Rightarrow \overline{x_s}^{\overline{p}} \setminus (\overline{t} + \overline{u})$ if $\{\!|\overline{x_s}^{\overline{p}} \setminus \overline{t}|\!\} \neq \emptyset$

(P6) $\overline{x_s}^{\overline{p}} \setminus \overline{t} \Rightarrow \bot$ if $\{\!|\overline{x_s}^{\overline{p}} \setminus \overline{t}|\!\} = \emptyset$

Figure 2 $\mathcal{R}_p$: reduce pattern of the form $t \times p$;
$\overline{v}, \overline{v_1}, \ldots, \overline{v_n}$, $\overline{w}, \overline{w_1}, \ldots, \overline{w_n}$ range over quasi-additive patterns, $\overline{u}, \overline{t}$ range over pure regular additive patterns, $\overline{t_1}, \ldots, \overline{t_n}$ range over pure symbolic patterns, $\overline{p}, \overline{q}$ range over constructor patterns, $\overline{x}$ ranges over pattern variables. α, β expand to all the symbols in $\mathcal{C}$, δ expands to all symbols in $\mathcal{C}^{n>0}$.

with an empty ground semantics is itself empty, while rule $S1$ expresses the distributivity of conjunction over cartesian products. Similarly, rules $S2$ and $S3$ express the distributivity of conjunction over disjunction.

The semantics of a variable of a given sort is the set of all ground constructor patterns of the respective sort. Thus, the difference between the ground semantics of any pattern and the ground semantics of a variable of the same sort is the empty set (rule $M1$). We should emphasize that $\overline{x_s}$ is a variable ranging over pattern variables at the object level and that z_i are fresh pattern variables seen as constants at the TRS level (*i.e.* $\overline{x_s}$ matches any z_{i_s}). The rules $M2 - M6$ correspond to set operation laws for complements. Rule $M7$ corresponds to the set difference of cartesian products; the case when the head symbol is a constant c corresponds to the rule $c \setminus c \Rightarrow \bot$. Rule $M8$ corresponds just to the special case where complemented sets are disjoint.

Similarly, the rules $T1 - T2$ indicate that the intersection with the set of all terms has no effect, rule $T3$ corresponds to distribution laws for the joint intersection, while $T4$ corresponds to the disjointed case.

The ground semantics of a variable is obtained by considering for each constructor of the

appropriate sort the set of all terms having this symbol at the root position and taking the union of all these sets (see Definition 3.4). The original TRS presented a rule encoding this behaviour but we have now to take into account the variable annotations and this would ultimately result into a rule of the form $\overline{x}^{s}_{-\overline{p}} \setminus \overline{q} \Rightarrow \sum_{c \in \mathcal{C}_s} c({x_1}^{s_1}_{-\overline{p}}, \ldots, {x_m}^{s_m}_{-\overline{p}}) \setminus \overline{p} \setminus \overline{q}$, which, as mentioned earlier, would lead to non termination.

Instead of this rule, $\mathcal{R}_p$ uses rule P1 which is specific to the left side of a conjunction. The rules P2, P3 and P4 expresses the respective behaviour of conjunction over complements $(A \cap (B \setminus C) = (A \setminus C) \cap B = (A \cap B) \setminus C)$. We can now observe that, thanks to the initial form of the conjunction $t \times p$ to be reduced and to the preservation by $\mathcal{R}_p$ of the *regular* property of a pattern, terms on the right of the symbol $\times$ are always be *regular*. Thus, by applying rule T1, $\mathcal{R}_p$ will always end up reducing the number of variables on the right hand side of the conjunctions $t \times p$. In other words, terms on the right of $\times$ will never be extended and this intuitively guarantees the termination of the reduction.

Finally, we can observe that, thanks to the algorithm introduced in Figure 1, we can determine if $\{\!\{\overline{x}^{s}_{-\overline{p}} \setminus \overline{v}\}\!\} = \emptyset$. Moreover, by definition, $\{\!\{t\}\!\} = \emptyset$ if and only if $[\![t]\!] = \emptyset$. Therefore, the TRS is finalized by the rule P6, which eliminates (when possible) annotated variables. Note that, in order to apply P6 exhaustively, $\mathcal{R}_p$ also needs a rule to perform some $\setminus$-factorization around variables, resulting in the rule P5.

The TRS $\mathcal{R}_p$ obtained is proved to provide classical rewriting guarantees: convergence and preservation of the semantics.

▶ **Lemma 4.7** (Convergence). *The rewriting system $\mathcal{R}_p$ is confluent and terminating.*

▶ **Proposition 4.8** (Ground semantics preservation). *For any extended patterns p, q, if $p \twoheadrightarrow_{\mathcal{R}_p} q$ then $[\![p]\!] = [\![q]\!]$.*

While we cannot provide a simple description of the normal forms obtained by reduction of a general extended patterns, $\mathcal{R}_p$ can be used to establish the emptyness of a given intersection:

▶ **Proposition 4.9.** *Given a quasi-additive pattern t and a constructor pattern p, we have $t \times p \twoheadrightarrow_{\mathcal{R}_p} \bot$ if and only if $[\![t \times p]\!] = \emptyset$*

Thanks to the above proposition and to the TRS $\mathcal{R}_p$ which allows one to check if a conjuction reduces to $\bot$, it is possible to systematically verify pattern-free properties for any term in $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

5 Semantics preservation for CTRS

Pattern-free properties rely on the symbol annotations and assume thus a specific shape for the normal forms of reducible terms. This assumption should be checked by verifying that the CTRSs defining the annotated symbols are consistent with these annotations, *i.e.* verifying that the semantics is preserved by reduction.

▶ **Definition 5.1** (Semantics preservation). *A rewrite rule $l \to r$ is* semantics preserving *iff $[\![r]\!] \subseteq [\![l]\!]$. A TRS is semantics preserving iff all its rewrites rules are.*

In particular, in the context of pattern eliminating transformation, we have CTRS with rules of the form $f^{-p}(l_1, \ldots, l_n) \to r$. Therefore, since the semantics of the left-hand side of the rewrite rule is the set of p-free values then such a rule is *semantics preserving* if and only if its right-hand side is p-free, *i.e.* $f^{-p}(l_1, \ldots, l_n) \to r$ is *semantics preserving* iff r is p-free.

The properties of the ground semantics thus ensure that semantics preservation of a CTRS carries over to the induced rewriting relation:

▶ **Proposition 5.2** (Semantics preservation). *Given a semantics preserving CTRS $\mathcal{R}$ we have*

$\forall t, v \in \mathcal{T}(\mathcal{F}),$ *if* $t \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}} v,$ *then* $[\![v]\!] \subseteq [\![t]\!]$.

As an immediate consequence we obtain the preservation of the pattern-free properties:

▶ **Corollary 5.3** (Pattern-free preservation). *Given a semantics preserving CTRS $\mathcal{R}$ we have*

$\forall t, v \in \mathcal{T}(\mathcal{F}), p \in \mathcal{T}(\mathcal{C}, \mathcal{X}),$ *if* t *p-free and* $t \longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}} v,$ *then* v *p-free.*

▶ **Example 5.4.** Given the signature from Example 3.6, we consider the following CTRS:

$$\begin{array}{rcl@{\qquad}rcl} f(c_1(x,y)) & \twoheadrightarrow & c_1(g(x), f(y)) & g(c_4(z)) & \twoheadrightarrow & c_3(c_2(z)) \\ f(c_2(z)) & \twoheadrightarrow & c_2(z) & g(c_3(y)) & \twoheadrightarrow & c_3(f(y)) \end{array}$$

We have seen in Example 4.6 that $r_1 = c_1(g(x), f(y))$ is p_1-free. Similarly, using $\mathcal{R}_p$, we can check that $r_2 = c_3(f(y))$ is p_2-free.

Moreover, s_3 excludes both p_1 and p_2, hence $c_2(z_{s_3})$ is p_1-free and $c_3(c_2(z_{s_3}))$ is p_2-free (which could again be established through $\mathcal{R}_p$).

Thus the CTRS is semantics preserving, meaning that for all $v_1 \in \mathcal{T}_{s_1}(\mathcal{C})$, $f(v_1)$ is and stays p_1-free through every reduction step, and for all $v_2 \in \mathcal{T}_{s_2}(\mathcal{C})$, $g(v_2)$ is and stays p_2-free through every reduction step.

Thus, given the method presented in Section 4, we can check that the right-hand sides of all rules verify the pattern-free property corresponding to the annotation of the head defined symbol of the left-hand side. We can therefore verify that a given CTRS is sound *w.r.t.* the chosen annotations.

▶ **Example 5.5.** Let's consider the flattening example presented in the introduction. We define a signature $\Sigma = (\mathcal{S}, \mathcal{F})$ with $\mathcal{S} = \{Expr, List\}$ and $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$ where $\mathcal{C} = \{int : Int \mapsto Expr, str : String \mapsto Expr, lst : List \mapsto Expr, nil : [] \mapsto List, conc : Expr \times List \mapsto List\}$, with Int and String being builtin sorts, and $\mathcal{D} = \{flattenE : Expr \mapsto Expr, flattenL : List \mapsto List, concat : List \times List \mapsto List\}$. Since we aim at eliminating the pattern $p = conc(lst(l_1), l_2)$, the defined symbols $flattenE$ and $flattenL$ are annotated with the pattern p.

The corresponding functions can be implemented through the following CTRS:

$$\left\{\begin{array}{lcl} flattenE(str(s)) & \twoheadrightarrow & str(s) \\ flattenE(lst(l)) & \twoheadrightarrow & lst(flattenL(l)) \\ flattenL(nil()) & \twoheadrightarrow & nil() \\ flattenL(conc(str(s), l)) & \twoheadrightarrow & conc(str(s), flattenL(l)) \\ flattenL(conc(lst(l_1), l_2)) & \twoheadrightarrow & flattenL(concat(l_1, l_2)) \\ concat(conc(e, l_1), l_2) & \twoheadrightarrow & conc(e, concat(l_1, l_2)) \\ concat(nil(), l) & \twoheadrightarrow & l \end{array}\right.$$

Thanks to the method introduced in Section 4 we can check that the right-hand sides of the first 5 rules are p-free and hence, that the CTRS is semantics preserving. Consequently, according to Proposition 5.3, we know that any term with a defined symbol head $flattenE$ or $flattenL$ is p-free and stays p-free through the reduction. This CTRS is clearly terminating and complete and thus, we can guarantee that the normal form of such terms are p-free values.

6 Related work

While the work presented in this paper presents an original approach to express and ensure a particular category of syntactical guarantees associated to program transformation, a number of different approaches presenting methods to obtain some guarantees for similar classes of functions exist in the literature.

Tree automata completion Tree automata completion consists in techniques used to compute an approximation of the set of terms reachable by a rewriting relation [14]. Such techniques could therefore be applied to solve similar problems to the one presented in this paper. The application of this approach is nevertheless usually conditioned by the termination of both the TRS and the set of equational approximations used [29, 15]. Thus, while providing a more precise characterization of the approximations of the normal forms, these techniques are constrained by these termination conditions. Therefore we believe the formalism presented in this paper provides a viable and original alternative to such techniques, particularly in the context of verification of pass transformations [20].

Recursion schemes Some formalism propose to deal with higher order functions through the use of higher order recursion schemes, a form of higher order grammars that are used as generators of (possibly infinite) trees [21]. In such approaches the verification problems are solved by model checking the recursion schemes generated from the given functional program. Higher order recursion scheme have also been extended to include pattern matching [26] and provide the basis for automatic abstraction refinement. These techniques address in a clever way the control-flow analysis of functional programs while the formalism proposed in our work is more focused on providing syntactic guarantees on the shape of the tree obtained through a pass-like transformation. The use of the annotation system also contributes to a more precise way to express and control the considered over-approximation.

Tree transducers Besides terms rewriting systems, another popular approach for specifying transformations consists in the use of tree transducers [22]. Transducers have indeed been shown to have a number of appealing properties when applied for strings, even infinite [2], and most notably can provide an interesting approach for model checking certain classes of programs thanks to the decidability of general verification problems [1]. Though the verification problems we tackle here are significantly more strenuous for tree transducers, Kobayashi et al. introduced in [22] a class of higher order tree transducers which can be modeled by recursion schemes and thus, provided a sound and complete algorithm to solve verification problems over that class. We claim that annotated TRSs are easier to grasp when specifying pass-like transformations and are less intrusive for expressing the pattern-free properties.

Refinement types Formalisms such as refinement types [11] can be seen as an alternative approach for verifying the absence, or presence, of specific patterns. In particular, notions such as constructor subtypes [6] could be used to construct complex type systems whose type checking would provide guarantees similar to the ones provided by our formalism. This would however result in the construction of multiple type systems in order to type check each transformation as was the case in the original inspiration of our work [20].

7 Conclusion and perspectives

We have proposed a method to statically analyse constructor term rewrite systems and verify the absence of patterns from the corresponding normal forms. We can thus guarantee not only that some constructors are not present in the normal forms but we can also be more specific and verify that more complex constructs cannot be retrieved in the result of the

reduction. Such an approach avoids the burden of specifying an intermediate language for each of the so-called passes in a compilation process, the user being just requested to indicate the patterns that should be eliminated by the respective transformation.

Different termination analysis techniques [3, 17, 16] and corresponding tools like AProVE [12] and TTT2 [23] can be used for checking the termination of the rewriting systems before applying our method for checking pattern-free properties. On the other hand, our approach does not rely on the termination of the analyzed rewriting systems; in case the rewriting systems is not terminating a final value is not obtained but the intermediate terms in the infinite reduction verify nevertheless the pattern-free properties *w.r.t.* the specified annotations.

We believe this formalism opens a lot of opportunities for further developments. In particular, this method could be extended in the context of automatic rewriting rule generation techniques, such as the one introduced in [8], in order to implement transformation approaches of passes such as in [20]. An implementation is under development.

References

1 Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 599–610. ACM, 2011. doi:10.1145/1926385.1926454.

2 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

3 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

4 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

5 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*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007. doi:10.1007/978-3-540-73449-9_5.

6 Gilles Barthe and Maria João Frade. Constructor subtyping. In *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99*, volume 1576 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 1999. doi:10.1007/3-540-49099-X_8.

7 Françoise Bellegarde. Program transformation and rewriting. In *Rewriting Techniques and Applications, 4th International Conference, RTA-91*, volume 488 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 1991. doi:10.1007/3-540-53904-2_99.

8 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*, volume 36 of *LIPIcs*, pages 74–88. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.RTA.2015.74.

9 Horatiu Cirstea and Pierre-Etienne Moreau. Generic encodings of constructor rewriting systems. In *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019*, pages 8:1–8:12. ACM, 2019. doi:10.1145/3354166.3354173.

10 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. The maude 2.0 system. In *Rewriting Techniques and Applications, 14th International Conference, RTA 2003*, volume 2706 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2003. doi:10.1007/3-540-44881-0_7.

11 Timothy S. Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI)*, pages 268–277. ACM, 1991. doi:10.1145/113445.113468.

12 Carsten Fuhs, Jürgen Giesl, Michael Parting, Peter Schneider-Kamp, and Stephan Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automatic Reasoning*, 47(2):133–160, 2011. doi:10.1007/s10817-010-9215-9.

13 Jacques Garrigue. Programming with polymorphic variants. In *In ACM Workshop on ML*, 1998.

14 Thomas Genet. Towards static analysis of functional programs using tree automata completion. In *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014*, volume 8663 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2014. doi:10.1007/978-3-319-12904-4_8.

15 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.

16 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automatic Reasoning*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

17 Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1-2):172–199, 2005. doi:10.1016/j.ic.2004.10.004.

18 Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 257–321. The MIT Press, 1991.

19 Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986. doi:10.1137/0215084.

20 Andrew W. Keep and R. Kent Dybvig. A nanopass framework for commercial compiler development. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13*, pages 343–350. ACM, 2013. doi:10.1145/2500365.2500618.

21 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 20099*, pages 416–428. ACM, 2009. doi:10.1145/1480881.1480933.

22 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*, pages 495–508. ACM, 2010. doi:10.1145/1706299.1706355.

23 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean termination tool 2. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21.

24 David Lacey and Oege de Moor. Imperative program transformation by rewriting. In Reinhard Wilhelm, editor, *Compiler Construction, 10th International Conference, CC 2001*, volume 2027 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2001. doi:10.1007/3-540-45306-7_5.

25 José Meseguer and Christiano Braga. Modular rewriting semantics of programming languages. In *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004*, volume 3116 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 2004. doi:10.1007/978-3-540-27815-3_29.

26 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*, pages 587–598. ACM, 2011. doi:10.1145/1926385.1926453.

27 François Pottier. Visitors unchained. *Proceedings of the ACM on Programming Languages*, 1(ICFP):28:1–28:28, 2017. doi:10.1145/3110272.

28 Grigore Rosu and Traian-Florin Serbanuta. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010. doi:10.1016/j.jlap.2010.03.012.

29 Toshinori Takai. A verification technique using term rewriting systems and abstract interpretation. In *Rewriting Techniques and Applications, 15th International Conference, RTA 2004*, volume 3091 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2004. doi:10.1007/978-3-540-25979-4_9.

30 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds.

31 Eelco Visser. Strategic pattern matching. In *Rewriting Techniques and Applications, 10th International Conference, RTA-99*, volume 1631 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 1999. doi:10.1007/3-540-48685-2_3.

A Proofs

▶ **Proposition 3.5.** $\forall t \in \mathcal{T}(\mathcal{F},\mathcal{X}), [\![t]\!] = [\![\tilde{t}]\!]$

Proof. The proof is immediate by induction on the form of t. ◀

▶ **Proposition 3.7.** *Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X}), p \in \mathcal{T}_\perp(\mathcal{C},\mathcal{X})$, t is p-free iff $\forall v \in [\![t]\!], v$ is p-free.*

Proof. If $t \in \mathcal{T}(\mathcal{C})$, then $[\![t]\!] = \{t\}$, hence the relation.

If $t = u\,[f^{-q}(t_1,\dots,t_m)]_\omega \in \mathcal{T}(\mathcal{F})$ with $f \in \mathcal{D}_s$, t is p-free iff $\forall v \in \mathcal{T}_s(\mathcal{C})$ q-free, $u\,[v]_\omega$ is p-free. We prove that $[\![t]\!] = \bigcup_{v\in[\![x_s^{-q}]\!]} [\![u\,[v]_\omega]\!]$. First if $\omega = \epsilon$, then $t = f^{-q}(t_1,\dots,t_m)$ thus $[\![t]\!] = [\![x_s^{-q}]\!] = \bigcup_{v\in[\![x_s^{-q}]\!]} [\![v]\!]$. Otherwise, we can prove it by induction on the form of u.

The base case is when $u = g^{-q'}(u_1,\dots,u_n)$ with $g \in \mathcal{D}_{s'}$. We have $\exists\, i \in [1,n]$ such that $\omega = i.\omega'$, hence $t = g^{-q'}(u_1,\dots,u_i\,[f^{-q}(t_1,\dots,t_m)]_{\omega'},\dots,u_n)$ and thus $[\![t]\!] = \left[\!\left[x_{s'}^{-q'}\right]\!\right]$ and $\forall v \in \mathcal{T}_s(\mathcal{C})$ q-free, $[\![u\,[v]_\omega]\!] = \left[\!\left[x_{s'}^{-q'}\right]\!\right]$. Hence $[\![t]\!] = \bigcup_{v\in[\![x_s^{-q}]\!]} [\![u\,[v]_\omega]\!]$.

For the induction, we then have the case when $u = c(u_1,\dots,u_n)$, with $c \in \mathcal{C}_{s'}$. We have $\exists\, i \in [1,n]$ such that $\omega = i.\omega'$, thus $t = c(u_1,\dots,u_i\,[f^{-q}(t_1,\dots,t_m)]_{\omega'},\dots,u_n)$ and $[\![t]\!] = \{c(v_1,\dots,v_n) \mid (v_1,\dots,v_n) \in [\![u_1]\!]\times\dots\times[\![u_i\,[f^{-q}(t_1,\dots,t_m)]_{\omega'}]\!]\times\dots\times[\![u_n]\!]\}$. Hence, by induction on u_i, $[\![t]\!] = \bigcup_{v\in[\![x_s^{-q}]\!]} [\![u\,[v]_\omega]\!]$.

Then a simple induction on the number of defined symbols in $t \in \mathcal{T}(\mathcal{F})$, gives us that t is p-free if and only if $\forall v \in [\![t]\!], v$ is p-free.

Finally, if $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, t is p-free iff $\forall\sigma$ such that $\sigma(t) \in \mathcal{T}(\mathcal{F})$, $\sigma(t)$ is p-free. Moreover, we can prove by induction on the form of t that $[\![t]\!] = \{u \mid \exists\,\sigma, u \in [\![\sigma(t)]\!]\} = \bigcup_\sigma [\![\sigma(t)]\!]$. Then $\forall\sigma, \sigma(t) \in \mathcal{T}(\mathcal{F})$, hence the property. ◀

▶ **Proposition 4.2** (Pattern-free vs Deep Semantics)**.** *Let $p \in \mathcal{T}(\mathcal{C},\mathcal{X}), t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, t is p-free iff $\{\!\{\tilde{t}\}\!\} \cap [\![p]\!] = \emptyset$.*

Proof. By definition, $\{\!\{\tilde{t}\}\!\} = \{u_{|\omega} \mid u \in [\![\tilde{t}]\!], \omega \in \mathcal{P}os\,(u)\} = \{u_{|\omega} \mid u \in [\![t]\!], \omega \in \mathcal{P}os\,(u)\}$, thus $\{\!\{\tilde{t}\}\!\} \cap [\![p]\!] = \emptyset$ if and only if $\forall u \in [\![t]\!], \omega \in \mathcal{P}os\,(u), p \not\leq u_{|\omega}$, *i.e.* $\forall u \in [\![t]\!], u$ is p-free. Therefore, thanks to Proposition 3.7, t is p-free if and only if $\{\!\{\tilde{t}\}\!\} \cap [\![p]\!] = \emptyset$. ◀

▶ **Proposition 4.3.** *For any pattern $p \in \mathcal{T}_\perp(\mathcal{C},\mathcal{X})$, constructor symbol $c \in \mathcal{C}$ such that $c : s_1\times\dots\times s_n \mapsto s$, and terms $(t_1,\dots,t_n) \in \mathcal{T}_{s_1}(\mathcal{C},\mathcal{X}^a)\times\dots\times\mathcal{T}_{s_n}(\mathcal{C},\mathcal{X}^a)$, we have:*

$$\{\!\{c(t_1,\dots,t_n)\}\!\} = [\![c(t_1,\dots,t_n)]\!] \cup \left(\bigcup_{i=1}^{n} \{\!\{t_i\}\!\}\right)$$

Proof. We consider the inclusions in the 2 directions separately. If $t \in \{\!\{c(t_1,\dots,t_n)\}\!\}$, then $\exists\, u \in [\![c(t_1,\dots,t_n)]\!], \omega \in \mathcal{P}os\,(u)$ such that $t = u_{|\omega}$. If $\omega = \epsilon$, then $t \in [\![c(t_1,\dots,t_n)]\!]$, otherwise $\omega = i.\omega'$ with $i \in [1,n]$ and therefore $\exists\, u' \in [\![t_i]\!]$ such that $t = u'_{|\omega'}$, *i.e.* $t \in \{\!\{t_i\}\!\}$. Hence the direct inclusion. For the indirect inclusion, we can first remark that any $v \in [\![c(t_1,\dots,t_n)]\!]$ is also in $\{\!\{c(t_1,\dots,t_n)\}\!\}$. Let's now consider $v \in \{\!\{t_i\}\!\}$ for some i, *i.e.* $\exists\, u_i \in [\![t_i]\!], \omega \in \mathcal{P}os\,(u_i)$ such that $v = u_{i|\omega}$, then for all $u = c(u_1,\dots,u_n)$ such that $\forall j \neq i, u_j \in [\![t_j]\!]$, $v = u_{|i.\omega}$, *i.e.* $v \in \{\!\{c(t_1,\dots,t_n)\}\!\}$. Hence the indirect inclusion. ◀

In order to prove Proposition 4.5, we introduce some auxiliary notions and prove some intermediate results.

First we analyse the deep semantics $\{\!\{x_s^{-p} \setminus r\}\!\}$. We can observe that $[\![x_s^{-p} \setminus r]\!] \subseteq [\![x_s \setminus (r+p)]\!]$, therefore if the latter is empty so is the first. Otherwise, we know that none of

the patterns in $(r+p)$ is x_s. Moreover, for any $c, c' \in \mathcal{C}_s$, we have $[\![c(p_1,\ldots,p_n) \setminus c(q_1,\ldots,q_n)]\!] = [\![\sum_{i=1}^{n} c(p_1,\ldots,p_i \setminus q_i,\ldots,p_n)]\!]$ and $[\![c(p_1,\ldots,p_n) \setminus c'(q_1,\ldots,q_m)]\!] = [\![c(p_1,\ldots,p_n)]\!]$. Therefore, given c, r and p we can construct a set denoted $Q_c(r+p)$ of n-tuples $q = (q_1,\ldots,q_n)$, with n the arity of c, by successively distributing the patterns of $r+p$ that have the given constructor c as head (denoted $r^1,\ldots,r^k$), as follows:

$$
\begin{aligned}
[\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus (r+p)]\!] &= [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus (r^1+\ldots+r^k)]\!] \\
&= [\![(c(x_{s_1}^{-p} \setminus \bot,\ldots,x_{s_n}^{-p} \setminus \bot) \setminus c(r_1^1,\ldots,r_n^1)) \setminus (r^2+\ldots+r^k)]\!] \\
&= \left[\!\!\left[\left(\sum_{i\in[1,n]} c(x_{s_1}^{-p} \setminus \bot,\ldots,x_{s_i}^{-p} \setminus r_i^1,\ldots,x_{s_n}^{-p} \setminus \bot) \right) \setminus (c(r_1^2,\ldots,r_n^2)+\ldots+r^k) \right]\!\!\right] \\
&= \left[\!\!\left[\left(\sum_{j\in[1,n]} \sum_{i\in[1,n]} c(x_{s_1}^{-p} \setminus \bot,\ldots,x_{s_j}^{-p} \setminus r_j^2,\ldots,x_{s_i}^{-p} \setminus r_i^1,\ldots,x_{s_n}^{-p} \setminus \bot) \right) \setminus (r^3+\ldots+r^k) \right]\!\!\right] \\
&\quad \vdots \\
&= \left[\!\!\left[\sum_{q\in Q_c(r+p)} c(x_{s_1}^{-p} \setminus q_1,\ldots,x_{s_n}^{-p} \setminus q_n) \right]\!\!\right]
\end{aligned}
$$

Thus as shown in development (2), we get:

$$
\{\!\{x_s^{-p} \setminus r\}\!\} = \begin{cases} \emptyset & \text{if } [\![x_s \setminus (r+p)]\!] = \emptyset \vee \forall c \in \mathcal{C}_s, Q'_c(r+p) = \emptyset \\ [\![x_s^{-p} \setminus r]\!] \;\cup \bigcup\limits_{c\in\mathcal{C}_s} \bigcup\limits_{q\in Q'_c(r+p)} \bigcup\limits_{i\in[1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\} & \text{else} \end{cases}
$$

with $Q'_c(u) = \{q \mid q \in Q_c(u) \wedge \forall i \in [1,n], [\![x_{s_i}^{-p} \setminus q_i]\!] \neq \emptyset\}$.

Given $Q_c(u)$ and $Q'c(u)$ defined above, we can now introduce the following abstractions for the deep semantics:

▶ **Definition A.1.** *Let $p \in \mathcal{T}_\bot(\mathcal{C},\mathcal{X})$, a couple (s,r), with $s \in \mathcal{S}$ and r a sum of constructor patterns such that $r = \bot \vee r : s$, and S a finite set of such couples,*

- $$\lfloor x_s^{-p} \setminus r \rfloor = \begin{cases} \emptyset & \text{if } [\![x_s \setminus (r+p)]\!] = \emptyset \vee \forall c \in \mathcal{C}_s, Q'_c(r+p) = \emptyset \\ \{(s,r)\} \;\cup \bigcup\limits_{c\in\mathcal{C}_s} \bigcup\limits_{q\in Q'_c(r+p)} \bigcup\limits_{i\in[1,n]} \lfloor x_{s_i}^{-p} \setminus q_i \rfloor & \text{else} \end{cases}$$

  Such that, as shown above, we have $\{\!\{x_s^{-p} \setminus r\}\!\} = \bigcup_{(s',p')\in\lfloor x_s^{-p}\setminus r\rfloor} [\![x_{s'}^{-p} \setminus p']\!]$ and $\lfloor x_s^{-p} \setminus r \rfloor = \emptyset$ iff $[\![x_s^{-p} \setminus r]\!] = \emptyset$. Thus $Q'_c(u) = \{q \mid q \in Q_c(u) \wedge \forall i \in [1,n], \lfloor x_{s_i}^{-p} \setminus q_i \rfloor \neq \emptyset\}$.

- $$\lfloor x_s^{-p} \setminus r \rfloor_S = \begin{cases} \emptyset & \text{if} \quad [\![x_s \setminus (r+p)]\!] = \emptyset \\ S & \text{else if} \quad \exists (s,r') \in S, [\![r']\!] = [\![r]\!] \\ \emptyset & \text{else if} \quad \forall c \in \mathcal{C}_s, Q_c^{S\cup\{(s,r)\}}(r+p) = \emptyset \\ \{(s,r)\} \;\cup \bigcup\limits_{c\in\mathcal{C}_s} \bigcup\limits_{q\in Q_c^{S\cup\{(s,r)\}}(r+p)} \bigcup\limits_{i\in[1,n]} \lfloor x_{s_i}^{-p} \setminus q_i \rfloor_{S\cup\{(s,r)\}} & \text{else} \end{cases}$$

  with $Q_c^S(u) = \{q \mid q \in Q_c(u) \wedge \forall i \in [1,n], \lfloor x_s^{-p} \setminus r \rfloor_S \neq \emptyset\}$

Moreover, we note $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} = \bigcup\limits_{(s',p')\in(\lfloor x_s^{-p}\setminus r\rfloor_S \setminus S)} [\![x_{s'}^{-p} \setminus p']\!]$

▶ **Lemma A.2.** *Let $p \in \mathcal{T}_\bot(\mathcal{C},\mathcal{X})$, a couple (s,r), with $s \in \mathcal{S}$ and r a sum of constructor pattern such that $r = \bot$ or $r : s$, and S a finite set of such couples. We have $\lfloor x_s^{-p} \setminus r \rfloor \neq \emptyset$ if and only if $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset \neq \emptyset$*

Proof. We consider the following (possibly infinite) tree structure: $N = \langle (s',p'), c, q, L\rangle$ such that $c : s_1 \times \ldots \times s_n \mapsto s' \in \mathcal{C}, q \in Q_c(p'+p)$ and $L = [N_1,\ldots,N_n]$ with $\forall i, N_i$ is of the form $\langle (s_i,r_i), [\ldots]\rangle$ with $[\![r_i]\!] = [\![q_i]\!]$. We remark that $Q_c(p'+p)$ is correctly defined if and only $[\![x_{s'} \setminus (p'+p)]\!] \neq \emptyset$, then if such a tree exists, we have, for all nodes $\langle (s',p'), [\ldots]\rangle, [\![x_{s'} \setminus (p'+p)]\!] \neq \emptyset$.

As $[\![r]\!] = [\![r']\!]$ implies $[\![x_s^{-p} \setminus r]\!] = [\![x_s^{-p} \setminus r']\!]$, by construction, if $\{\!\{x_s^{-p} \setminus r\}\!\} \neq \emptyset$, then there exists such a tree. And conversely, if there exists such a tree, as for all nodes $\langle (s', p'), [\ldots]\rangle, [\![x_{s'} \setminus (p' + p)]\!] \neq \emptyset$, then we can construct a term t by assigning to each node the value of the constructor label, such that, by construction, $t : s$ and t is p-free, hence $t \in [\![x_s^{-p} \setminus r]\!]$, and thus $\{\!\{x_s^{-p} \setminus r\}\!\} \neq \emptyset$.

We prove now that $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset \neq \emptyset$ if and only if there exists such a tree. If there exists such tree, we can prove that for each node $N = \langle (s', p'), [\ldots]\rangle$ of this tree $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$ with S the set of pairs (ζ, ρ) of each node in the path from the root of the tree to N. If the tree is finite, this is obviously true for each leaf. Otherwise, there is at least one infinite branch, and as each p' is a sum of a subterm r and subterms of p, there is only a finite number of such terms with a different ground semantics (as $[\![u + u]\!] = [\![u]\!]$). Hence, for each infinite branch, there is a node $N = \langle (\zeta, \rho), [\ldots]\rangle$ such that the path from the root of the tree to N contains a node $\langle (\zeta, \rho'), [\ldots]\rangle$ with $[\![\rho]\!] = [\![\rho']\!]$, hence $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S$. We can then prove by induction that this holds for each node. Thus, we have, for the root node, $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset \neq \emptyset$. If $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset \neq \emptyset$, by construction of $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset$, we can build a tree such that for each node $N = \langle (s', p'), [\ldots]\rangle$ of this tree $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$ with S the set of pairs (ζ, ρ) of each node in the path from the root of the tree to N, with each branch of the tree terminating on a node $\langle (\zeta, \rho), c, [\ldots]\rangle$ such that c is of arity 0 or there exists a node $\langle (\zeta, \rho'), [\ldots]\rangle$ with $[\![\rho]\!] = [\![\rho']\!]$ in which case we can repeat infinitely the path between the 2 nodes to get the desired tree.

Thus we have $\lfloor x_s^{-p} \setminus r \rfloor \neq \emptyset$ if and only if $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset \neq \emptyset$. ◀

▶ **Lemma A.3.** *Let $p \in \mathcal{T}_\perp(\mathcal{C}, \mathcal{X})$, a couple (s, r), with $s \in \mathcal{S}$ and r a sum of constructor patterns such that $r = \perp \vee r : s$, and S a finite set of such couples. We have:*

1. *If $\lfloor x_s^{-p} \setminus r \rfloor_S \neq \emptyset$ then $\forall (s', p')$ with $q : s$ or $q = \perp$, $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup \{(s', p')\}} \neq \emptyset$;*
2. *$\forall (s', p')$ with $p : s'$ or $p = \perp$, if $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup \{(s', p')\}} \neq \emptyset$ then $\forall S'$ such that $\lfloor x_{s'}^{-p} \setminus p' \rfloor_{S'} \neq \emptyset$, $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup S'} \neq \emptyset$;*
3. *$\forall (s', p')$, with $p : s'$ or $p = \perp$, if $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$, then $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s', p')\}} \subseteq \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S}$;*
4. *$\forall (s', p')$, with $p : s'$ or $p = \perp$, if $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$, then $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} \subseteq \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s', p')\}} \cup \{\!\{x_{s'}^{-p} \setminus p'\}\!\}_{\setminus S}$.*

Proof. We consider $P(u)$ the set of all patterns that we can construct by sum of subterms of u and p, and $P_{\setminus S}(u) = \{t \mid t \in P(u) \wedge \forall (s', p') \in S, [\![t]\!] \neq [\![p']\!]\}$. Thanks to this, we can prove the 4 properties by induction on S and r, such that $P_{\setminus S}(r)$ is strictly decreasing.

The base case is for S and r such that $\exists (s, r') \in S$ with $[\![r]\!] = [\![r']\!]$, in this case $\lfloor x_s^{-p} \setminus r \rfloor_S = S$, and thus all 4 properties hold. Let now S be a set such that $\forall (s', r') \in S, s \neq s' \vee [\![r]\!] \neq [\![r']\!]$, we suppose that $\forall c \in \mathcal{C}_s, \forall q \in Q_c(r + p)$, the properties hold for all s_i, q_i and $S \cup \{(s, r)\}$, and we want to prove that they hold for S and r. Indeed, as q_i is a sum of subterms of r and p, we have $P(q_i) \subseteq P(r)$, and we have $r \in P_{\setminus S}(r)$ but $r \notin P_{\setminus S \cup \{(s,r)\}}(q_i)$, hence $P_{\setminus S \cup \{(s,r)\}}(q_i) \subset P_{\setminus S}(r)$.

1. If $\lfloor x_s^{-p} \setminus r \rfloor_S \neq \emptyset$ then $\forall (s', p')$ with $q : s$ or $q = \perp$, $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup \{(s', p')\}} \neq \emptyset$;
   If $s' = s$ and $[\![p']\!] = [\![r]\!]$, then the property is obviously true. Otherwise, as $\lfloor x_s^{-p} \setminus r \rfloor_S \neq \emptyset$, we know that $[\![x_s \setminus (r + p)]\!] \neq \emptyset$ and that $\exists c \in \mathcal{C}_s$ such that $Q_c^{S \cup \{(s,r)\}}(r + p) \neq \emptyset$, *i.e.* $\exists q \in Q_c(r + p)$ such that $\forall i \in [1, n], \lfloor x_{s_i}^{-p} \setminus q_i \rfloor_{S \cup \{(s,r)\}} \neq \emptyset$. We can then apply the inductive property for $S \cup \{(s, r)\}$, hence $\forall i \in [1, n], \lfloor x_{s_i}^{-p} \setminus q_i \rfloor_{S \cup \{(s,r) \cup \{(s', p')\}} \neq \emptyset$, and thus $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup \{(s', p')\}} \neq \emptyset$.
2. $\forall (s', p')$ with $p : s'$ or $p = \perp$, if $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup \{(s', p')\}} \neq \emptyset$ then $\forall S'$ such that $\lfloor x_{s'}^{-p} \setminus p' \rfloor_{S'} \neq \emptyset$, $\lfloor x_s^{-p} \setminus r \rfloor_{S \cup S'} \neq \emptyset$;
   We proceed the exact same way.

3. $\forall (s', p')$, with $p : s'$ or $p = \bot$, if $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$, then $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s',p')\}} \subseteq \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S}$;
   If $s' = s$ and $[\![p']\!] = [\![r]\!]$, then the property is obviously true. Otherwise, as $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$, we have, thanks to the previous 2 properties, $\forall (s'', p'')$, $\lfloor x_{s''}^{-p} \setminus p'' \rfloor_{S \cup \{(s,r)\}} \neq \emptyset$ if and only if $\lfloor x_{s''}^{-p} \setminus p'' \rfloor_{S \cup \{(s,r),(s',p')\}} \neq \emptyset$. Thus, for all $c \in \mathcal{C}_s, Q_c^{S \cup \{(s',p'),(s,r)\}}(r+p) = Q_c^{S \cup \{(s',p')\}}(r+p)$. Therefore, if one is empty, so is the other, and both semantics then verify the property. Finally, if neither $[\![x_s \setminus (r+p)]\!]$ nor all $Q_c^S(r+p)$ are empty, we have:

$$\begin{aligned}
\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s',p')\}} &= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{S \cup \{(s,r),(s',p')\}}(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r),(s',p')\}} \\
&= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{S \cup \{(s,r)\}}(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r),(s',p')\}} \\
&\subseteq [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{S \cup \{(s,r)\}}(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \text{ by induction}
\end{aligned}$$

   And so $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s',q)\}} \subseteq \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S}$.
4. $\forall (s', p')$, with $p : s'$ or $p = \bot$, if $\lfloor x_{s'}^{-p} \setminus p' \rfloor_S \neq \emptyset$, then $\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} \subseteq \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S \cup \{(s',p')\}} \cup \{\!\{x_{s'}^{-p} \setminus p'\}\!\}_{\setminus S}$;
   We procede the exact same way.

◀

▶ **Proposition 4.5** (Correctness). *Let $s \in \mathcal{S}, p \in \mathcal{T}_\bot(\mathcal{C}, \mathcal{X})$ and $r : s$ a sum of constructor patterns, if $R = \mathit{getReachable}(s, p, \emptyset, r)$, then*

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

Moreover, we have $\{\!\{x_s^{-p} \setminus r\}\!\} = \emptyset$ iff $R = \emptyset$.

Proof. If $\lfloor x_s^{-p} \setminus r \rfloor_S \neq \emptyset$, thanks to Lemma A.3, we have:

$$\begin{aligned}
\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} &= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{S \cup \{(s,r)\}}(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \\
&= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \\
&\subseteq [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S}
\end{aligned}$$

and:

$$\begin{aligned}
\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} &= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^{S \cup \{(s,r)\}}(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \\
&= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \\
&= [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} (\{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S \cup \{(s,r)\}} \cup \{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S}) \\
&\supseteq [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S}
\end{aligned}$$

Therefore, we then have:

$$\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus S} = [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q_c^S(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus S}$$

And thus, for $S = \emptyset$ and since $\lfloor x_s^{-p} \setminus r \rfloor_\emptyset = \emptyset \iff [\![x_s^{-p} \setminus r]\!] = \emptyset$:

$$\{\!\{x_s^{-p} \setminus r\}\!\}_{\setminus \emptyset} = \begin{cases} \emptyset \quad \text{if } [\![x_s \setminus (r+p)]\!] = \emptyset \vee \forall c \in \mathcal{C}_s, Q'_c(r+p) = \emptyset \\ [\![x_s^{-p} \setminus r]\!] \cup \bigcup_{c \in \mathcal{C}_s} \bigcup_{q \in Q'_c(r+p)} \bigcup_{i \in [1,n]} \{\!\{x_{s_i}^{-p} \setminus q_i\}\!\}_{\setminus \emptyset} \quad \text{else} \end{cases}$$

This relation is equivalent to the one for $[\![x_s^{-p} \setminus r]\!]$, hence: $[\![x_s^{-p} \setminus r]\!]_{\setminus \emptyset} = [\![x_s^{-p} \setminus r]\!]$

Finally, by looking at the algorithm, we can observe that $getReachable(s, p, S, r) = \lfloor x_s^{-p} \setminus r \rfloor_S$. To do so, we reference each **return** case of the algorithm by $(R1), (R2), (R3)$ and $(R4)$, in order of appearance.

The algorithm starts by conflating r with $r + p$ when $p : s$ (otherwise, p has no effect), thanks to the first *if* of the algorithm. Thanks to the second *if* we then have an empty **return** on $(R1)$ when $[\![x_s \setminus r]\!] = \emptyset$. And the third *if* leads to **returning** S on $(R2)$ when $\exists\,(s, r') \in S, [\![r']\!] = [\![r]\!]$.

If the algorithm did not **return** on $(R1)$ or $(R2)$, we then have, with the conflated r, $[\![x_s^{-p} \setminus r]\!] = \left[\!\!\left[\sum_{c \in \mathcal{C}_s} c(x_{s_1}^{-p}, \ldots, x_{s_n}^{-p}) \setminus r\right]\!\!\right]$. Thus the algorithm loops on $c \in \mathcal{C}_s$. The first nested *for* loop computes the set $Q_c(r)$ obtained, as mentioned, by successively distributing the patterns of r that have the given constructor c as head. The second *for* loop then recursively calls `getReachable` on the couples (s_i, q_i) obtained this way, and updates R with the results obtained.

At this point, as the algorithm did not **return** on $(R1)$ we have $[\![x_s^{-p} \setminus r]\!] = \emptyset$ if and only if, $\forall c \in \mathcal{C}_s, Q'_c(r) = \emptyset$, hence the boolean variable *reachable* that stays false when $\forall c \in \mathcal{C}_s, Q_c^{S \cup \{(s,r)\}}(r) = \emptyset$, resulting in an empty **return** $(R4)$. Similarly, $[\![c(x_{s_1}^{-p} \setminus q_1, \ldots, x_{s_n}^{-p} \setminus q_n)]\!]$ is empty if and only if $\exists\, i$ such that $[\![x_{s_i}^{-p} \setminus q_i]\!] = \emptyset$, so R is updated with the result of the recursive calls for a given $c \in \mathcal{C}_s$ and $q \in Q_c(r)$ only if none of these recursive calls returns an empty result. We thus have the concatenated result as described in the definition of $\lfloor x_s^{-p} \setminus r \rfloor_S$ on **return** $(R3)$.

◀

▶ **Lemma 4.7** (Convergence). *The rewriting system $\mathcal{R}_p$ is confluent and terminating.*

Proof. A meta-encoding of a complete approximation of the rule schema $\mathcal{R}_p$ is provided in Appendix B. Automatic termination proof tools such as TTT2 and AProVE have been used to prove that this meta-encoding is terminating and we can thus directly conclude to the termination of $\mathcal{R}_p$

We show the local confluence of the system by proving that all critical pairs induced by rewrite rules of the system converge. We have the following critical pairs:
$(A1) - (A2)$ (converge directly),
$(A1) - (S1)$ and $(A2) - (S1)$ (converge with $E1$ and $A1/A2$),
$(A1) - (S2)$ and $(A2) - (S2)$ (converge with $E2$ and $A1/A2$),
$(A1) - (S3)$ and $(A2) - (S3)$ (converge with $E3$ and $A1/A2$),
$(A1) - (M3)$ and $(A2) - (M3)$ (converge with $M5$ and $A1/A2$),
$(A1) - (M6)$ and $(A2) - (M3)$ (converge with $M2$),
$(E1) - (M6)$ (converge with $M5$ and $E1$, twice $M5$),
$(E1) - (M7)$ only left possible (converge with $M5$ and $M2$, n times $E1$, n times $A1/A2$),
$(E1) - (M8)$ left (converge with $M5$ and $E1$),
$(E1) - (M8)$ right (converge with $M2$),
$(E2) - (E3)$ (converge directly),
$(E2) - (S3)$ and $(E3) - (S2)$ (converge with twice $S2/S3$, $A1/A2$),
$(E2) - (T1)$ and $(E3) - (T2)$ (converge directly),
$(S1) - (M6)$ (converge with $M3$, twice $M6$ and $S1$, twice $M3$),
$(S1) - (M7)$ only left possible (converge with $M3$, twice $M7$ and $M3$, n times $S1$),
$(S1) - (M8)$ left (converge with $M3$, twice $M8$ and $S1$),
$(S1) - (M8)$ right (converge with $M6$, twice $M8$),
$(S1) - (T3)$ left (converge with $S2$, twice $T3$ and $S2$, $S1$),

$(S1) - (T3)$ right (converge with $S3$, twice $T3$ and $S3$, $S1$),
$(S1) - (T4)$ left (converge with $S2$, twice $T4$, $A1/A2$),
$(S1) - (T4)$ right (converge with $S3$, twice $T4$, $A1/A2$),
$(S1) - (P1)$ (converge with $S3$, twice $P1$ and $S1$, $M3$, $S3$),
$(S2) - (S3)$ (converge with $S3$ and $S2$),
$(S2) - (T1)$ (converge with twice $T1$),
$(S3) - (P4)$ (converge with twice $P4$ and $S3$, twice $M6$),
$(S3) - (T2)$ (converge with twice $T1$),
$(M1) - (M3)$ (converge with twice $M1$, $A1/A2$),
$(M1) - (M5)$ (converge directly),
$(M1) - (P6)$ (converge directly),
$(M2) - (M3)$ (converge with twice $M1$),
$(M2) - (M5)$ (converge directly),
$(T1) - (T2)$ (converge directly),
$(T1) - (P4)$ (converge with $T1$),
$(T2) - (P3)$ (converge with $T2$). ◀

▶ **Proposition 4.8** (Ground semantics preservation). *For any extended patterns p, q, if $p \twoheadrightarrow_{\mathcal{R}_p} q$ then $[\![p]\!] = [\![q]\!]$.*

Proof. We prove that the ground semantics of the left-hand side and right-hand side of the rewrite rules of $\mathcal{R}_p$ are the same.

In the case of the rule (E1), as we have $[\![\delta(p_1, \ldots, p_n)]\!] = \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![p_1]\!] \times \ldots \times [\![p_n]\!]\}$ and the ground semantics of $\bot$ is empty, so is the semantics of $\delta(v_1, \ldots, \bot, \ldots, v_n)$. Hence the equality of ground semantics of the 2 sides of the rule. For the rule (S1), we have:

$$\begin{array}{rcl} [\![\delta(v_1, \ldots, v_i + w_i, \ldots, v_n)]\!] & = & \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![v_1]\!] \times \ldots \times [\![v_i + w_i]\!] \times \ldots \times [\![v_n]\!]\} \\ & = & \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![v_1]\!] \times \ldots \times [\![v_i]\!] \cup [\![w_i]\!] \times \ldots \times [\![v_n]\!]\} \\ & = & \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![v_1]\!] \times \ldots \times [\![v_i]\!] \times \ldots \times [\![v_n]\!]\} \\ & & \cup \{\delta(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![v_1]\!] \times \ldots \times [\![w_i]\!] \times \ldots \times [\![v_n]\!]\} \\ & = & [\![\delta(v_1, \ldots, v_i, \ldots, v_n)]\!] \cup [\![\delta(v_1, \ldots, w_i, \ldots, v_n)]\!] \end{array}$$

For rules (M7) and (T3), we consider both inclusion separately:

(M7): If $v \in [\![\alpha(v_1, \ldots, v_n) \setminus \alpha(t_1, \ldots, t_n)]\!]$, then $v \in [\![\alpha(v_1, \ldots, v_n)]\!]$ and $v \notin [\![\alpha(t_1, \ldots, t_n)]\!]$. As $[\![\alpha(p_1, \ldots, p_n)]\!] = \{\alpha(w_1, \ldots, w_n) \mid (w_1, \ldots, w_n) \in [\![p_1]\!] \times \ldots \times [\![p_n]\!]\}$, $v = \alpha(w_1, \ldots, w_n)$ such that $\forall i \in [1, n], w_i \in [\![v_i]\!]$ and $\exists\, j \in [1, n], w_j \in [\![t_j]\!]$. Therefore, $w_j \in [\![v_j]\!] \setminus [\![t_j]\!]$ and thus $v \in [\![\alpha(v_1, \ldots, v_j \setminus t_j, \ldots, v_n)]\!]$, and finally $v \in \left[\!\!\left[\sum_{k \in [1,n]} \alpha(v_1, \ldots, v_k \setminus t_k, \ldots, v_n)\right]\!\!\right]$. Hence the first inclusion. We can show that if $v \in \left[\!\!\left[\sum_{k \in [1,n]} \alpha(v_1, \ldots, v_k \setminus t_k, \ldots, v_n)\right]\!\!\right]$, then $v \in [\![\alpha(v_1, \ldots, v_n) \setminus \alpha(t_1, \ldots, t_n)]\!]$ similarly in order to prove the second inclusion.

(T3): If $v \in [\![\alpha(v_1, \ldots, v_n) \times \alpha(w_1, \ldots, w_n)]\!]$, then $v \in [\![\alpha(v_1, \ldots, v_n)]\!]$ and $v \in [\![\alpha(w_1, \ldots, w_n)]\!]$. As $[\![\alpha(p_1, \ldots, p_n)]\!] = \{\alpha(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in [\![p_1]\!] \times \ldots \times [\![p_n]\!]\}$, $v = \alpha(t_1, \ldots, t_n)$ such that $\forall i \in [1, n], t_i \in [\![v_i]\!]$ and $t_i \in [\![w_i]\!]$. Therefore, $\forall i \in [1, n], t_i \in [\![v_i]\!] \cap [\![w_i]\!]$ and thus $v \in [\![\alpha(v_1 \times w_1, \ldots, v_n \times w_n)]\!]$. Hence the first inclusion. Similarly, we can show that if $v \in [\![\alpha(v_1 \times w_1, \ldots, v_n \times w_n)]\!]$, then $v \in [\![\alpha(v_1, \ldots, v_n) \times \alpha(w_1, \ldots, w_n)]\!]$, to prove the second inclusion.

For the rest of the rules but (P1), the definition of ground semantics of extended patterns and properties of set operations give us the equality between the ground semantics of 2 side of each rule in a fairly straightforward manner. In particular, in the case of rules (M1), (T1) and (T2), we can remark that, as we only consider well-sorted extended patterns, in these 3 rules we have $v : s$ and therefore $[\![v]\!] \subseteq [\![x_s^{-\bot}]\!]$. Hence the equality of ground semantics of the 2 sides of these rules.

Finally, for (P1), we first prove that:

$$[\![x_s^{-p}]\!] \quad = \quad \bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!] \tag{1}$$

Let's consider both inclusion separately. Let $t \in \bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!]$, *i.e.* $\exists\, c \in \mathcal{C}_s$ such that $t \in [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p})]\!]$ and $t \notin [\![p]\!]$. Thus $\exists\,(t_1,\ldots,t_n) \in [\![x_{s_1}^{-p}]\!] \times \ldots [\![x_{s_n}^{-p}]\!]$ such that $p \not\preccurlyeq t = c(t_1,\ldots,t_n)$. Therefore, $t : s$ and $\forall\omega \in \mathcal{P}os\,(t)\,, p \not\preccurlyeq t_{|\omega}$, *i.e.* t is p-free. Hence $t \in [\![x_s^{-p}]\!]$. Let $t \in [\![x_s^{-p}]\!]$, then $t : s$ and t is p-free. Thus $\exists\, c : s_1 \times \ldots \times s_n \mapsto s, (t_1,\ldots,t_n) \in \mathcal{T}_{s_1}(\mathcal{C}) \times \ldots \times \mathcal{T}_{s_n}(\mathcal{C})$ such that $t = c(t_1,\ldots,t_n)$ with $\forall i \in [1,n], t_i$ is p-free. Therefore, $t \in [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p})]\!]$ and, as $p \not\preccurlyeq t$, $t \notin [\![p]\!]$. Hence $t \in [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!]$, and finally $t \in \bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!]$.

Therefore, we have $[\![x_s^{-p}]\!] = [\![\sum_{c\in\mathcal{C}_s} c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!]$ hence

$$\begin{aligned}
[\![x_s^{-p} \times \alpha(v_1,\ldots,v_n)]\!] &= [\![\left(\textstyle\sum_{c\in\mathcal{C}_s} c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p\right) \times \alpha(v_1,\ldots,v_n)]\!] \\
&= \left(\bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \setminus p]\!]\right) \cap [\![\alpha(v_1,\ldots,v_n)]\!] \\
&= \left(\bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p})]\!] \setminus [\![p]\!]\right) \cap [\![\alpha(v_1,\ldots,v_n)]\!] \\
&= \bigcup_{c\in\mathcal{C}_s} [\![c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p})]\!] \cap \left([\![\alpha(v_1,\ldots,v_n)]\!] \setminus [\![p]\!]\right) \\
&= \left[\!\!\left[\sum_{c\in\mathcal{C}_s} c(x_{s_1}^{-p},\ldots,x_{s_n}^{-p}) \times \left(\alpha(v_1,\ldots,v_n) \setminus p\right)\right]\!\!\right]
\end{aligned}$$

◀

In order to prove Proposition 4.9, we introduce the notion of *depth* of an extended pattern and we show Lemmas A.5, A.6 and A.7.

▶ **Definition A.4** (Depth). *We define the notion of* depth *of an extended pattern:*

- $depth(x_s^{-p}) = depth(x_s^{-p} \setminus w) = depth(\bot) = 1$
- $depth(c(t_1,\ldots,t_n)) = 1 + max_{i\in[1,n]}(depth(t_i))$
- $depth(t_1 + t_2) = max(depth(t_1), depth(t_2))$
- $depth(t_1 \setminus t_2) = depth(t_1)$

▶ **Lemma A.5.** *If t is a (quasi-)additive pattern, then $t\downarrow_{\mathcal{R}_p}$ is a (quasi-)additive pattern such that $depth(t\downarrow_{\mathcal{R}_p}) \leq depth(t)$.*

Moreover the normal form $t\downarrow_{\mathcal{R}_p} = v$ is either $\bot$ or a pure term such that $\forall\omega_+ \in \mathcal{P}os\,(v)\,, v(\omega_+) = + \Longrightarrow \forall\omega < \omega_+, v(\omega) = +$ and $[\![t]\!] = \emptyset$ if and only if $v = \bot$.

Proof. We can first observe that the only rule that apply on an additive pattern are A1, A2, E1 and S1 (and the same rules plus M1 and P6 for quasi-additive patterns). Moreover, for each rule, it reduces a (quasi-)additive pattern into a (quasi-)additive pattern. Therefore, the normal form of an (quasi-)additive pattern, is indeed an (quasi-)additive pattern.

Moreover, the *depth* measure induces a monotonic ordering over quasi-additive patterns with regard to the $\leq$ operators, *i.e.* $depth(u) \leq depth(v)$ implies $depth(t\,[u]_\omega) \leq depth(t\,[v]_\omega)$. Finally, as the *depth* is decreasing on all applicable rules, we know that $depth(t\downarrow_{\mathcal{R}_p}) \leq depth(t)$.

Let's now suppose that $v = t\downarrow_{\mathcal{R}_p}$ contains a sum below a constructor, *i.e.* contains a subterm of the form $c(v_1,\ldots,v_i + u_i,\ldots,v_n)$, which would be a redex for S1, and thus v would not be a normal form. Therefore, v does not contain a sum below a constructor.

Finally, if $t\downarrow_{\mathcal{R}_p}=\bot$ then Proposition 4.8 ensures that $[\![t]\!]=[\![\bot]\!]=\emptyset$. We note $v=t\downarrow_{\mathcal{R}_p}$, once again we know that $[\![t]\!]=[\![v]\!]$, let's prove that if $[\![v]\!]=\emptyset$ then $v=\bot$. We suppose that $v\neq\bot$ and we prove by induction that v is not in normal form.

If $v=x_s^{-p}\setminus u$ and $[\![v]\!]=\emptyset$, then rule P6 applies, thus v is not in normal form. If $v=v_1+v_2$, then $[\![v_1]\!]=[\![v_2]\!]=\emptyset$, thus by induction, either $v_1=v_2=\bot$ in which case both A1 and A2 applies, or at least one of them is not in normal form. In both cases, v is not in normal form. Finally, if $v=c(v_1,\dots,v_n)$, then $\exists\, i\in[1,n]$ such that $[\![v_i]\!]=\emptyset$, thus by induction, either $v_i=\bot$ and rule E1 applies or v_i is not in normal form. In both cases, v is not in normal form.

Therefore, if $v\neq\bot$ is a quasi-additive such that $[\![v]\!]=\emptyset$, then v is not in normal form. Thus, if $[\![t]\!]=\emptyset$, then $t\downarrow_{\mathcal{R}_p}=\bot$. ◀

▶ **Lemma A.6.** *If t a quasi-additive pattern and u a regular additive pattern, then $v=(t\setminus u)\downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern such that $depth(v)\leq depth(t\setminus u)$.*

Proof. We can remark that $\mathcal{R}_p$ preserves the *regular* property of a pattern, thus, thanks to Lemma A.5, we can suppose, by confluence, that t and u are in normal form, we then prove this lemma by induction on the form of t and u.

In the case when $u=x_s^{-\bot}$, rule M1 applies to $t\setminus u$ and reduces it to $v=\bot$ which cannot be reduced furthermore. Moreover $depth(\bot)=1\leq depth(t\setminus u)$. In the case when $u=\bot$, rule M2 applies to $t\setminus u$ and reduces it to t, and as we supposed t in normal form, it cannot be reduced anymore. Moreover $depth(t)=depth(t\setminus u)$.

In other cases, we proceed by induction:

- if $u=u_1+u_2$, we have to consider the different form of t. If $t=x_s^{-p}$, the only rule that can apply is P6, which, if it does, reduces $t\setminus u$ to $\bot$, thus, either way, the term v obtained cannot be reduced furthermore and is quasi-additive pattern with $depth(v)=1=depth(t\setminus u)$. If $t=x_s^{-p}\setminus u'$, the only rules that can apply are P5 and P6. If P5 applies, it reduces $t\setminus u$ to $x_s^{-p}\setminus(u'+u)$ for which only P6 can apply. If P6 applies, it reduces $t\setminus u$ to $\bot\setminus u$, which is then reduced to $\bot$ by M5, and $x_s^{-p}\setminus(u'+u)$ to $\bot$. In all cases, the term v obtained cannot be reduced furthermore and is quasi-additive pattern with $depth(v)=1=depth(t\setminus u)$. If $t=c(t_1,\dots,t_n)$, rule M6 applies to $t\setminus u$ and reduces it to $(t\setminus u_1)\setminus u_2$. Moreover, by induction on u, $v'=(t\setminus u_1)\downarrow_{\mathcal{R}_p}$ and $v=(v'\setminus u_2)\downarrow_{\mathcal{R}_p}$ are both quasi-additive patterns such that $depth(v)\leq depth(v')\leq depth(t\setminus u)$. Hence, by confluence, $v=(t\setminus u)\downarrow_{\mathcal{R}_p}$. Finally, if $t=t_1+t_2$, then rule M3 applies to $t\setminus u$ and reduces it to $(t_1\setminus u)+(t_2\setminus u)$. Moreover, by induction on t, $v_1=(t_1\setminus u)\downarrow_{\mathcal{R}_p}$ and $v_2=(t_2\setminus u)\downarrow_{\mathcal{R}_p}$ are both quasi-additive patterns such that $depth(v_1)\leq depth(t\setminus u)$ and $depth(v_2)\leq depth(t\setminus u)$. Therefore, $depth(v_1+v_2)\leq depth(t\setminus)u$, and $t\setminus u\longrightarrow\!\!\!\!\!\rightarrow_{\mathcal{R}_p} v_1+v_2$. So as v_1+v_2 is a quasi-additive pattern we know, thanks to Lemma A.5, that $v=(v_1+v_2)\downarrow_{\mathcal{R}_p}=(t\setminus u)_{\mathcal{R}_p}$ is a quasi-additive pattern such that $depth(v)\leq depth(v_1+v_2)\leq depth(t\setminus u)$.
- if $u=c(u_1,\dots,u_n)$, with $\forall i\in[1,n], u_i$ a regular symbolic pattern (as u is in normal form), we have to consider the different form of t. If $t=x_s^{-p}$, the only rule that can apply is P6, which, if it does, reduces $t\setminus u$ to $\bot$, thus, either way, the term v obtained cannot be reduced furthermore and is quasi-additive pattern with $depth(v)=1=depth(t\setminus u)$. If $t=x_s^{-p}\setminus u'$, the only rules that can apply are P5 and P6. If P5 applies, it reduces $t\setminus u$ to $x_s^{-p}\setminus(u'+u)$ for which only P6 can apply. If P6 applies, it reduces $t\setminus u$ to $\bot\setminus u$, which is then reduced to $\bot$ by M5, and $x_s^{-p}\setminus(u'+u)$ to $\bot$. In all cases, the term v obtained cannot be reduced furthermore and is quasi-additive pattern with $depth(v)=1=depth(t\setminus u)$. For the case when $t=c'(t_1,\dots,t_m)$, if $c\neq c'$, rule M8 applies to $t\setminus u$ and reduces it to t. Otherwise rule M7 applies to $t\setminus u$ and reduces it to $\sum_{i\in[1,n]} c(t_1,\dots,t_i\setminus$

$u_i, \ldots, t_n)$, and by induction on t, $\forall i, v_i = (t_i \setminus u_i) \downarrow_{\mathcal{R}_p}$ a quasi-additive pattern such that $depth(v_i) \leq depth(t_i \setminus u_i)$. Therefore, $t \setminus u \twoheadrightarrow_{\mathcal{R}_p} \sum_{i\in[1,n]} c(t_1, \ldots, v_i, \ldots, t_n)$ and by monotonicity, $depth(\sum_{i\in[1,n]} c(t_1, \ldots, v_i, \ldots, t_n)) \leq depth(t \setminus u)$. Moreover, $w = \sum_{i\in[1,n]} c(t_1, \ldots, v_i, \ldots, t_n)$ is a quasi-additive pattern, so according to Lemma A.5, $v = w \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern such that $depth(v) \leq depth(w)$. Hence, by confluence, $v = (t \setminus u) \downarrow_{\mathcal{R}_p}$ and $depth(v) \leq depth(t \setminus u)$. Finally, if $t = t_1 + t_2$, we proceed identically as for $u = u_1 + u_2$.

◀

▶ **Lemma A.7.** *If t a quasi-additive pattern and u a regular quasi-additive pattern, then $(t \times u) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern.*

Proof. We can remark that $\mathcal{R}_p$ preserves the *regular* property of a pattern, thus, thanks to Lemma A.5, we can suppose, by confluence, that t and u are in normal form, we then prove this lemma by induction on the depth of u and the form of t and u.

The base case is for u such that $depth(u) = 1$. We proceed by induction on the form u such that $depth(u) = 1$. If $u = x_s^{-\perp}$, the only rule that applies to $t \setminus u$ is T2, which reduces it to t.

If $u = x_s^{-\perp} \setminus v$ with v a regular additive pattern, we proceed by induction on t. If $t = c(t_1, \ldots, t_n)$ or $t = x_s^{-p}$ or $t = x_s^{-p} \setminus w$ the only rules that applies to $t \setminus u$ are, respectively, P2 or P3 or P4, which with T1 reduces it to $t \setminus v$. And thanks to Lemma A.6 we know that $(t \setminus v) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern. Finally, if $t = t_1 + t_2$, then rule S2 applies to $t \setminus u$ and reduces it to $(t_1 \times u) + (t_2 \times u)$. Moreover, by induction on t_1 and t_2, $v_1 = (t_1 \times u) \downarrow_{\mathcal{R}_p}$ and $v_2 = (t_2 \times u) \downarrow_{\mathcal{R}_p}$ are both quasi-additive patterns. So, as $v_1 + v_2$ is a quasi-additive pattern, we know, thanks to Lemma A.5, that $(v_1 + v_2) \downarrow_{\mathcal{R}_p} = (t \setminus u) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern.

If $u = c()$, we proceed by induction on the form of t. If $t = c()$, then rule T3 applies to $t \times u$ and reduces it to $c()$. If $t = c'(t_1, \ldots, t_n)$ with $c' \neq c$, then rule T4 applies to $t \setminus u$ and reduces it to $\perp$. If $t = x_s^{-p}$, then rule P1 applies to $t \times u$ and reduces it to $\sum_{d\in\mathcal{C}_s} d(x_{s_1}^{-p}, \ldots, x_{s_n}^{-p}) \times (c() \setminus p)$. If $p = c()$, then rule M7 applies to $c() \setminus p$ and reduces it to $\perp$, leading to rule E2 applying and ultimately reducing $t \setminus u$ to $\perp$. If $p = c'(p_1, \ldots, p_n)$ with $c' \neq c$, then rule M8 applies to $c() \setminus p$ and reduces it to $c()$, and because we only consider well sorted extended patterns, $c \in \mathcal{C}_s$, thus repeatedly applying S3, T4, A1/ A2 and finally T3 ultimately reduces $t \setminus u$ to $c()$. Finally, if $t = t_1 + t_2$, we proceed exactly the same way as in the case when $u = x_s^{-\perp} \setminus v$ to prove the induction step on the form of t.

Finally, if $u = u_1 + u_2$ with $depth(u_1) = depth(u_2) = 1$, then rule S3 applies to $t \setminus u$ and reduces it to $(t \times u_1) + (t \times u_2)$. Moreover, by induction on $u1$ and u_2, $v_1 = (t \times u_1) \downarrow_{\mathcal{R}_p}$ and $v_2 = (t \times u_2) \downarrow_{\mathcal{R}_p}$ are both quasi-additive patterns. So as $v_1 + v_2$ is a quasi-additive pattern we know, thanks to Lemma A.5, that $(v_1 + v_2) \downarrow_{\mathcal{R}_p} = (t \setminus u)_{\mathcal{R}_p}$ is a quasi-additive pattern.

We now suppose $depth(u) = n > 1$ and for all quasi-additive pattern v such that $depth(v) < n$, for all quasi-additive pattern τ, $(\tau \setminus v) \downarrow_{\mathcal{R}_p}$ is quasi-additive pattern. Let's prove by induction on the form of t and u that, for all quasi-additive pattern $t, (t \setminus u) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern.

If $u = c(u_1, \ldots, u_m)$ with $\forall i \in [1, m], depth(u_i) < n$, we proceed by induction on the form of t. For the case when $t = c'(t_1, \ldots, t_{m'})$, if $c \neq c$, then rule T4 applies to $t \setminus u$ and reduces it to $\perp$. Otherwise, rule T3 applies to $t \times u$ and reduces it to $c(t_1 \times u_1, \ldots, t_n \times u_m)$. Moreover, by induction on the depth of u we know that $\forall i \in [1, m], v_i = (t_i \times u_i) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern, thus $c(v_1, \ldots, v_m)$ is a quasi-additive pattern and we know, thanks to Lemma A.5, that $c(v_1, \ldots, v_m) \downarrow_{\mathcal{R}_p} = (t \setminus u) \downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern. If $t = x_s^{-p}$, then rule T4 applies to $t \setminus u$ and reduces it to $\sum_{d\in\mathcal{C}_s} d(x_{s_1}^{-p}, \ldots, x_{s_{m'}}^{-p}) \times (u \setminus p)$. Thanks to

Lemma A.6, we know that $(u \setminus p)\downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern which depth is less than or equal to n, and we can easily show that its either $\perp$ or a sum of quasi-additive patterns of the form $c(w_1, \ldots, w_n)$, with $\forall i \in [1, m], depth(w_i) < n$. If $(u \setminus p)\downarrow_{\mathcal{R}_p} = \perp$, then rule E3 applies and thus $(t \times u)\downarrow \mathcal{R}_p = \perp$. Otherwise, by applying recursively S2/S3, T4/T3, and A1/A2 we get $t \times u \twoheadrightarrow_{\mathcal{R}_p} \sum c(x_{s_1}^{-p} \times w_1, \ldots, x_{s_m}^{-p} \times w_m)$. Moreover, by induction on the depth of u we know that $\forall i \in [1, m], v_i = (t_i \times w_i)\downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern, thus $\sum c(v_1, \ldots, v_m)$ is a quasi-additive pattern and we know, thanks to Lemma A.5, that $(\sum c(v_1, \ldots, v_m))\downarrow_{\mathcal{R}_p} = (t \setminus u)\downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern. Finally, if $t = t_1 + t_2$, we proceed exactly the same way as in the case when $u = x_s^{-\perp} \setminus v$ to prove the induction step on the form of t.

Finally, if $u = u_1 + u_2$, we proceed exactly the same way as in the case $depth(u) = 1$ to prove the induction step on the form u when $depth(u) > 1$. ◀

▶ **Proposition 4.9.** *Given a quasi-additive pattern t and a constructor pattern p, we have $t \times p \twoheadrightarrow_{\mathcal{R}_p} \perp$ if and only if $[\![t \times p]\!] = \emptyset$*

Proof. Based on Lemma A.7, we know that $(t \times p)\downarrow_{\mathcal{R}_p}$ is a quasi-additive pattern. Moreover, according to Proposition 4.8, the semantics of $t \times p$ is empty if and only if the semantics of its normal form is empty, hence, thanks to Lemma A.5, if and only if its normal form is $\perp$. ◀

▶ **Proposition 5.2** (Semantics preservation)**.** *Given a semantics preserving CTRS $\mathcal{R}$ we have*

$$\forall t, v \in \mathcal{T}(\mathcal{F}),\ \textit{if}\ t \twoheadrightarrow_{\mathcal{R}} v,\ \textit{then}\ [\![v]\!] \subseteq [\![t]\!]\,.$$

Proof. As all constructor rewrite rules are of the form $f(l_1, \ldots, l_n) \twoheadrightarrow r$, we have $\forall \sigma, [\![\sigma(r)]\!] \subseteq [\![r]\!] \subseteq [\![l]\!] = [\![\sigma(f(l_1, \ldots, l_n))]\!]$. Moreover, $\forall t \in \mathcal{T}(\mathcal{F}), \forall u, v \in \mathcal{T}(\mathcal{C})\ [\![v]\!] \subseteq [\![u]\!]$ implies $\forall \omega \in \mathcal{P}os\,(t)\ [\![t\,[v]_\omega]\!] \subseteq [\![t\,[u]_\omega]\!]$. Therefore, by definition of the rewriting relation induced by such a semantics preserving rule, we have $\forall u, v \in \mathcal{T}(\mathcal{F}),\ u \longrightarrow_{\mathcal{R}} v$ implies $[\![v]\!] \subseteq [\![u]\!]$. ◀

B Meta encoding of the rewriting system $\mathcal{R}_p$

The meta encoding of the rule schemas in Figure 2 is given below in a syntax usable by AProVE/TTT2. Both AProVE and TTT2 can be used to prove the termination of this rewriting system.

```
(VAR u u1 u2 v v1 v2 w f g lu lv l lacc n m i tail sig p q)
(RULES
  plus(bot,v) -> v
  plus(v,bot) -> v


  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)

  times(bot,v,sig) -> bot
  times(v,bot,sig) -> bot

  times(plus(u1,u2),v,sig) ->
        plus(times(u1,v,sig),times(u2,v,sig))
  times(v,plus(u1,u2),sig) ->
        plus(times(v,u1,sig),times(v,u2,sig))


  minus(v, var(n,bot)) -> bot
  minus(v, bot) -> v
  minus(plus(v1,v2), w) ->
        plus(minus(v1, w), minus(v2, w))

  minus(bot, appl(f,lv)) -> bot

  minus(appl(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,suc(i)) ->
        plus(genm7(f,lu,lv,i),
             appl(f,diff(lu,lv,suc(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(appl(f,lu), appl(g,lv)) -> appl(f,lu)


  times(v, var(n,bot), sig) -> v
  times(var(n,bot), v, sig) -> v

  times(appl(f,lu), appl(f,lv), sig) -> dist(appl(f,nil), prod(lu,lv,nil,sig)))
   prod(cons(u,lu),cons(v,lv),lacc,sig) -> prod(lu,lv,cons(times(u,v,sig),lacc),sig)

   dist(appl(f,l), prod(nil,nil,nil,sig)) -> appl(f,l)
   dist(appl(f,l), prod(nil,nil,cons(u,lu),sig)) -> dist(appl(f,cons(u,l)), prod(nil,nil,lu,sig))

  times(appl(f,lu), appl(g,lv), sig) -> bot

  times(var(n,p), appl(f,lv), sig) ->
         times(gensum(sig,p),minus(appl(f,lv), p), sig)
     gensum(nilsig,p) -> bot
     gensum(conssig(f,n,tail),p) ->
         plus(appl(f,genvar(n,p)), gensum(tail,p))
     genvar(z,p) -> nil
     genvar(s(n),p) -> cons(var(s(n),p),genvar(n,p))

  times(appl(f,lu), minus(var(m,p), t), sig) -> minus(times(appl(f,lu), var(m,p), sig), t)

  times(var(n,p), minus(var(m,q), t), sig) -> minus(times(var(n,p), var(m,q), sig), t)

   times(minus(var(n,p),v), t, sig) -> minus(times(var(n,p),v,sig), t)
   minus(minus(var(n,p),v), t) -> minus(var(n,p),plus(v,t))

   minus(var(n,p),v) -> bot
)
```