Classification, Extension and Applications of some Schematization Methods of Infinite Sets of Terms

Ali AMANISS
CRIN, INRIA-Lorraine, Nancy *

Abstract. *Schematization* is a method for finitely representing infinite sets of terms. It was introduced to find a solution for some inadequacies of systems based on first-order logic: divergence of rewriting systems, nontermination of some Prolog programs. The first method is *ω-terms*. It is extended by *I-terms*, *R-terms* and *primal grammars*. We call these methods *recurrent methods*. In this paper, we compare some classical schematization methods, namely *context-free grammars*, *set constraints* with *recurrent methods*. The result of this classification is that some methods are more expressive than others, regular grammars are appropriate to schematize regular languages (decidable properties), recurrent methods are appropriate to schematize non-regular languages (unification is decidable), no method can schematize both complicated regular and non-regular languages with interesting decidable properties. Here, we propose a method to achieve that with some tools (*matching, a method of automatically generating these schematizations*) and applications.

Keywords: Matching, Context-free grammars, Set Constraints, Schematization of Infinite Sets of Terms, Regular Languages.

1 Introduction

When we examine the rules deduced by the completion process of rewriting systems, the clauses deduced by a Prolog system, for some particular programs, we find a lot of structurally similar objects which cause the non-termination of the processes and gives a severe limitation to the practical use of this processes. There is, in general, an inductive relation, between the objects of these infinite sets, which allows us to schematize them finitely.

The first proposition to deal with this problems was given by H. Kirchner [10] who proposed the schematization of infinite sets of rules finitely using the notion of *meta-variable*. A meta-variable is a variable ranging on a recursive set of terms. If a meta-variable occurs in a term, we call it *meta-term* and if a rewrite rule is built over meta-terms, we call it *meta-rule*. This method has some problems. The first one is that unification between meta-terms is undecidable. The second one is that no systematic way of deriving meta-rules was given. But some heuristics for constructing meta-rules was proposed.

Chen and Hsiang proposed in [5, 6] another method, called ω-terms, for finitely representing infinite sets of terms. It consists of iterating terms with one hole along fixed path. The number of iterations is part of the syntax of the terms and may include integer variables. In [7], Comon gives a first generalization of this method allowing terms overlapping. He called his formalism *I-terms*. In [13], Salzer gives an extension of I-terms called *R-terms*. R-terms are a powerful class of schematization because they allow more than one hole for iterations. Hermann proposed a more powerful class of schematization [9].

Here, we give a complete comparison of these methods. We give a short presentation of *context-free grammars, set constraints* and we compare them with recurrent methods. From these comparisons we note that no method can schematize both complicated regular and non-regular languages with interesting decidable properties. Let us give an example from Logic Programming. Suppose that we have the following Prolog program.

* CRIN and INRIA-Lorraine, Campus Scientifique, BP 239, 54506 ,Vandoeuvre-lès-Nancy, France, e-mail: Amaniss@loria.fr

$$P(a).$$
$$P(f(x)) \leftarrow P(x).$$
$$P(g(x)) \leftarrow P(x).$$

The query $\leftarrow P(x)$. allows us to know all elements which verify P. It generates the following infinite set of clauses $\{P(a), P(f(a)), P(g(a)), P(f(g(a)), P(g(f(a)), \ldots\}$. This set is infinite and causes the nontermination of the SLD-resolution. Primal grammars, the powerful recurrent method, cannot deal with this example.

Here, we propose a method which allows us to schematize complicated regular and non-regular languages with some decidable properties and some applications.

2 Recurrent Methods

We call *recurrent methods* the classes of ω-terms [4], I-terms [7], R-terms [13], and primal grammars [9].

ω-terms, I-terms and R-terms use the notation $\Phi(h, N, l)$ where h and l are terms, N an integer variable and Φ an 3-ary functional symbol. $\Phi(h, N, l)$ is called a *generator*, and often denoted by Φ_N.

Example 1 The ω-term $\Phi(f(\diamond), N, a)$ schematizes the infinite set of terms $\{f^n(a) \mid n \in \mathcal{N}\}$, when the I-term $\Phi(f(\diamond), N, \Phi(g(\diamond), N, a))$, which is not an ω-term, schematizes the infinite set $\{f^n(g^n(a)) \mid n \in \mathrm{N}\}$.

Primal grammars are another class of schematization of infinite sets of first-order terms. This class is more powerful than the previous classes. It contains R-terms class. Its strength rests in the fact that we have the possibility to relate integer variables (counters) of a meta-terms. This allows us to do a dynamic instantiations for integer variables. With R-terms, instantiations of integer variables are done only statically and no relation can be built between different variables of an R-term. To define primal grammars[9], we begin with the following definitions.

The expression *integer variable* is replaced, for primal grammars, by *counter variable* or simply *counter*. In the ω-terms case, this variables are denoted by upper-case letters, for primal grammars we denote it by lower-case letters $(n, m, p, q, \ldots)$.

In addition to the signature $\mathcal{F}$ of *plain* symbols, let us consider another signature $\mathcal{H}$ of *definite symbols* such that $\mathcal{F} \cap \mathcal{H} = \emptyset$. Definite symbols will be denoted by a hat to distinguish them from the plain symbols. The arguments of a definite symbol $\widehat{f} \in \mathcal{H}$ are divided into two parts by semicolon (example $\widehat{f}(\vec{n}; \vec{t}))$. The part before the semicolon are called *counters*. Each definite symbol $\widehat{f}$ has a *counter arity*, denoted by $\beta(\widehat{f})$ indicating its number of counters. The first counter is called *principal counter*, other counters are called *secondary counters*.

Primal grammars use a special rewrite systems called *prime rewrite systems*.

3 Classification

The comparison of these methods in their expressness follows from definitions. We have ω-terms$\subset$I-terms $\subset$R-terms$\subset$primal grammar, see [2] and figure 1 for more information.

Now, we know that primal grammars are the powerful class of recurrent methods. In the following, we characterize each class inside primal grammars class. That is, we give necessary and sufficient conditions to obtain each class from primal grammars.

1. R-terms: a primal grammar is an R-term (i.e they represent the same infinite set of terms) if and only if each rewrite rule of this grammar is of the form:
   (a) $\widehat{f}(0; \vec{t}) \to r_1^{\widehat{f}}$ or
   (b) $\widehat{f}(s(n); \vec{t}) \to r_2^{\widehat{f}}[\widehat{f}(n; \vec{t})]_A$;

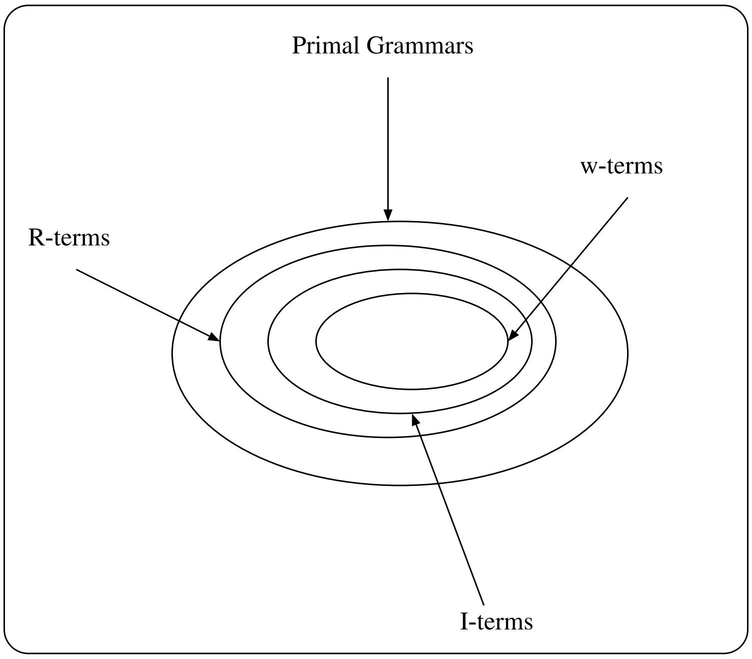

Fig. 1. Summary of Recurrent Methods Comparison.

where $\overrightarrow{t}$ is a vector of variables of $\widehat{r_1^f}$ and $\widehat{r_2^f}$. Hence, no secondary counter is needed. The vector of variables $\overrightarrow{t}$ is only used to have the condition $\mathcal{V}\mathrm{ar}(l) \subseteq \mathcal{V}\mathrm{ar}(r)$ for each rewrite rule $l \to r$.

Thus, R-terms class is primal grammars class where the rules verify these conditions;

2. I-terms: to have I-terms class, we take the same conditions that for R-terms with A reduced to a one iterated position;
3. ω-terms: ω-terms are obtained from primal grammars with the same conditions for I-terms but with $\widehat{r_1^f}$ and $\widehat{r_2^f}$ ground.

4 Tree Grammars and Recurrent Methods

In this section, we introduce trees grammars and compare them with recurrent methods.

A *context-free tree grammar* $G = (\Sigma, N, S, R)$ consists of disjoint signatures N (nonterminals) and Σ (terminals), a finite rewrite system R over $N \cup \Sigma$, and a distinct constant S of arity zero (axiom); all rules of R are of the form

$$A(x_1, \ldots, x_n) \to t$$

where A is of arity $n, x_1, \ldots, x_n$ are different variables, and $t \in \mathcal{T}(N \cup \Sigma, \{x_1, \ldots, x_n\})$.

G is *regular* if N contains only constant symbols (symbols of arity zero). G is said to be *top-context-free* if all proper right-hand sides of rules are in $\mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$; (they was introduced and called *grammaires corégulieres* in [3]

4.1 ω-terms and Tree Grammars

An ω-term t is called a *linear ω-term* if each integer variable appears at most once in t. For example, $f(x, x, \Phi(g(\diamond), N, a))$ is linear when $f(x, \Phi(g(\diamond), N, a), \Phi(h(\diamond), N, b))$ is not.

Proposition 1 Linear ω-terms class is a subclass of regular grammars.

Proof : See [1]$\triangleleft$

Example 2 Let $t = \Phi(f(\diamond), N, a)$. It is trivial that $\Omega(t) = \{f^n(a) \mid n \in \mathcal{N}\}$. This language is represented by the regular grammar with the following rules:

$$S \to a$$
$$S \to f(S)$$

but it is represented by the following top-context-free grammar, too:

$$\begin{array}{ll} S & \to A(a) \\ A(x) & \to A(f(x)) \\ A(x) & \to x \end{array}$$

Proposition 2 ω-terms class is a subclass of top-context-free grammars.

Proof : We give a construction proof. We construct a top-context-free grammar from each ω-term; see [1] for more information$\triangleleft$

See figure 2 for summary.

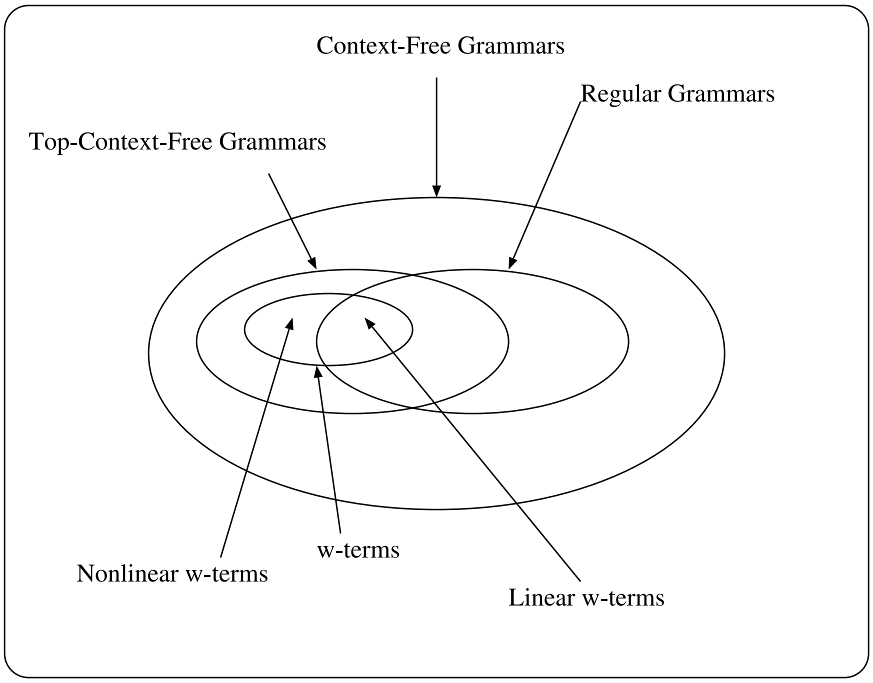

Fig. 2. ω-Terms and Tree Grammars.

4.2 I-terms and Tree Grammars

Here we give the relationship between I-terms and tree grammars. An I-term t is *linear I-term* if and only if each integer variable appears only once in t. For example, $f(x, x, \Phi(g(\diamond), N, a))$ is linear when $f(x, \Phi(h(\diamond), N, \Phi(g(\diamond), N, a)))$ is not. We call *flat I-terms* the subset of I-terms where the generators $t^N.u$ verify $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Proposition 3 Linear flat I-terms is a sub-class of regular tree grammars.

Proof : Let t be a flat linear I-term, we give a regular tree grammar G such that $L(G) = \Omega(t)$. We proceed by structural induction:

1. $t = x$, let G be the regular tree grammar $G = (\{x\}, \{A\}, A, \{A \to x\})$. It is trivial that $L(G) = \Omega(t)$.
2. $t = f(t_1, \ldots, t_n)$, by induction hypothesis there are grammars $G_i = (\Sigma_i, N_i, A_i, R_i)$ such that $\forall 1 \leq i \leq n, L(G_i) = \Omega(t_i)$. Let $r = A \to f(A_1, \ldots, A_n)$ and $G = (\bigcup_{i=1}^{n} \Sigma_i, \bigcup_{i=1}^{n} N_i \cup \{A\}, A, \bigcup_{i=1}^{n} R_i \cup \{r\})$. It is not difficult to prove that $L(G) = \Omega(t)$.
3. $t = \Phi(h[\diamond]_u, N, l)$, By induction hypothesis there is a grammar $G_l = (\Sigma_l, N_l, A_l, R_l)$ such that $L(G_l) = \Omega(l)$. Let $r_1 = A \to A_l, r_2 = A \to h[A]_u$ and $G = (\Sigma_l, N_l \cup \{A\}, A, R_l \cup \{r_1, r_2\})$. It is clear that $L(G) = \Omega(t)$ $\lhd$

Lemma 1 The language $L = \{\underbrace{f(f(\ldots f(}_{n} a, \underbrace{g^m(b)) \ldots, g^m(b)), g^m(b))}_{n} \mid n, m \in \mathcal{N}\}$ is not regular; but L is top-context-free.

Proof: See [1]$\lhd$

Example 3 *(Typical languages schematized by linear flat I-terms.)*We give some typical examples of construction of I-terms from linear regular grammars.

1. Let G be the grammar with axiom S and rules:

$$\begin{aligned} &S \to f(A, B) \\ &A \to g(A) \\ &A \to a \\ &B \to h(B) \\ &B \to b \end{aligned}$$

   It is clear that $L(G) = \{f(g^n(a), h^m(b) \mid n, m \in \mathcal{N}\}$, this language can be schematized by $t = f(\Phi(g(\diamond), N, a), \Phi(h(\diamond), M, b))$;
2. Let G be the grammar with axiom A and rules:

$$\begin{aligned} &A \to f(A) \\ &A \to B \\ &B \to g(B) \\ &B \to a \end{aligned}$$

   we have $L(G) = \{f^n(g^m(a)) \mid n, m \in \mathcal{N}\}$, this language can be schematized by $t = \Phi(f(\diamond), N, \Phi(g(\diamond), M, a))$.

Theorem 1 Linear I-terms class is a sub-class of top-context-free grammars which is not completely included in the regular grammars class.

Proof: We construct for each linear I-term a top-context-free grammar which schematizes the same set of terms. For complete proof see [1]. The above lemma gives the second part of the proof$\lhd$

Proof: grammar which schematizes the same set of terms. For complete proof see [1]. The following lemma give the second part of the proof$\lhd$

Definition 1 Let t be an I-term. We say that t is *semi-linear* if and only if

1. t is linear;
2. t is not linear and $t = \Phi(h[]_u, N, l)$ where h and l are semi-linear;
3. t is not linear and $t = t'[g_1, \ldots, g_n]$, t' is a term, $g_i = \Phi(h_i, N_i, l_i)$ are semi-linear generators, h_i and l_i have no integer variable in common and do not contain variables N_i.

Theorem 2 Semi-linear I-terms are a sub-class of top-context-free grammars.

Proof: See [1]◁

There are some languages that nonlinear I-terms can schematize but we conjecture that those languages are not context-free. An example of those languages is

$$L = \{\underbrace{f(\dots f(}_{n} b, \underbrace{g^n(a)) \dots, g^n(a))}_{n} \mid n, m \in \mathcal{N}\},$$

this language can be schematized by the nonlinear I-term $t = \Phi(f(\diamond, \Phi(g(\diamond), N, b)), N, a)$. See figure 3 for a summary of above comparisons.

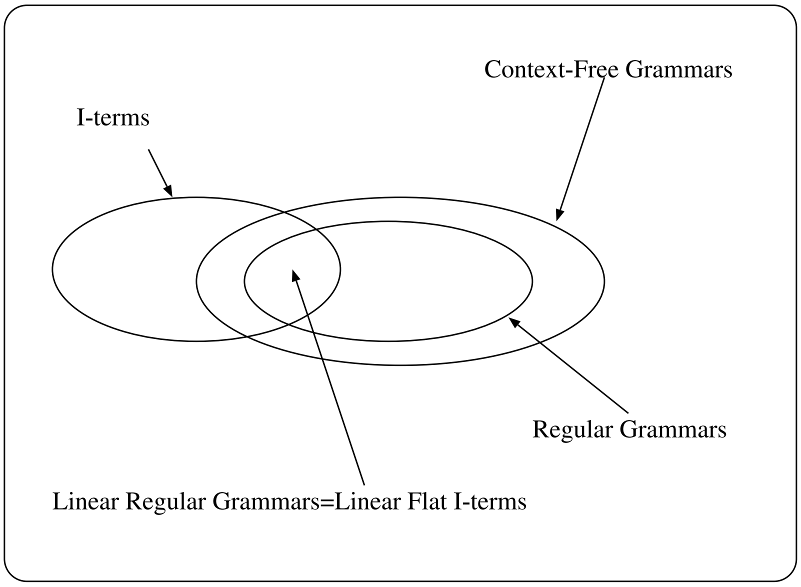

Fig. 3. I-Terms and Tree Grammars.

4.3 R-terms, Primal Grammars and Tree Grammars

R-terms and primal grammars use more than one iteration position. Thus, if in a term we have more than one iteration position for an integer variable then the language represented is not regular. Let t be the R-term $\Phi(f(\diamond, \diamond), N, a)$. We can naively represent it by the grammar G with the following rules $A \to a$ and $A \to f(A, A)$ but this grammar and t do not represent the same language, we have $\Omega(t) \subseteq L(G)$. In fact, $f(f(a, a), a) \in L(G)$ but $f(f(a, a), a) \notin \Omega(t)$.

If we define semi-linear R-terms as we do with I-terms we have the following result:

Theorem 3 Semi-linear R-terms are a sub-class of top-context-free grammars.

Proof: See [1]◁

5 Set Constraints and ω-terms

For an introduction to *set constraints* see [8]. Here we give a relationship between ω-terms and set constraints and we prove that linear ω-terms are a subclass of set constraints.

We say that a set constraints $s \supseteq t$ represents the same set of terms than an ω-term t if and only if $\Omega(t)$ is a least solution for some variable of $s \supseteq t$. Here we restrict set constraints to *definite set* *constraints* (set constraints of the form $a \supseteq t$ without complementation). It is known that if such set constraints, have a solution then they have a least solution. Further, this solution is regular in the sense that every variable is a regular set.

Proposition 4 Linear ω-terms is a subclass of definite set constraints.

Proof: It is sufficient to prove that for any linear ω-term t there is a collection of definite set constraints such that $\Omega(t)$ is the least solution for some variable of this set constraint.

If $t = a$ is a constant, the corresponding set constraint is $X \supseteq a$.

If $t = t'[\Phi(h_1[\diamond]_{u_1}, N_1, l_1)]_{p_1} \ldots [\Phi(h_1[\diamond]_{u_n}, N_n, l_n)]_{p_n}$ then for each generator Φ_{N_i} we give two set constraints:

$$X_i \supseteq l_i$$
$$X_i \supseteq h_i[X_i]_{u_i}$$

Then, we give the constraint:

$$Z \supseteq t'[X_1]_{p_1} \ldots [X_n]_{p_n}$$

$X_1, \ldots, X_n$ and Z are new variables$\triangleleft$

If we want to construct set constraints from nonlinear ω-terms, we need to have complementation in set constraints.

6 Extensions

From the previous comparisons, we note that no method can both schematize complicated regular and non-regular languages with interesting decidable properties.

6.1 Bounded Grammars

We use the notations of primal grammars [9] to introduce *bounded grammars*.

Definition 2 A *bounded grammar* G is $G = (\mathcal{F},\mathcal{D},\mathcal{R},t^*)$ where

for each symbol $\widehat{f} \in \mathcal{D}$, we have at most n rewrite rules of the form

$$\widehat{f}(0, \vec{m}; \vec{t}) \rightarrow r_1,$$

and at most n rewrite rules of the form :

$$\widehat{f}(s(n), \vec{m}; \vec{t}) \rightarrow r_2[\widehat{f}(n, \vec{m}; \vec{t})]_A$$

where A $\subset \mathcal{P}\mathrm{os}(r_2)$ and t^* is a primal term. We denote this class by $\mathcal{T}_5$.

This definition of grammars allows us to schematize some infinite sets that primal grammars cannot schematize. Let us give an example.

Example 4 Assume that $G = (\mathcal{F},\mathcal{H},\mathcal{R},t^*)$ where $\mathcal{F} = \{+, *, a, b\}$, $\mathcal{H} = \{\widehat{f}\}$, $t^* = \widehat{f}(n)$ and $\mathcal{R}$ contains the following rules,

$$\begin{aligned} \widehat{f}(0) &\rightarrow a \\ \widehat{f}(0) &\rightarrow b \\ \widehat{f}(s(n)) &\rightarrow \widehat{f}(n) + \widehat{f}(n) \\ \widehat{f}(s(n)) &\rightarrow \widehat{f}(n) * \widehat{f}(n) \end{aligned}$$

This grammar schematizes the infinite set $L(G) = \{a, b, a + a, a * a, (b + a) * (a + b), \ldots\}$

Proposition 5 Bounded grammars contains strictly primal grammars ($\mathcal{T}_4 \subset \mathcal{T}_5$)

Proof : The proof is trivial, because instead of taking several rewrite rules to define bounded grammars, we take only two rules to define primal grammars. Then we can deduce that $\mathcal{T}_4 \subset \mathcal{T}_5$. Now it remains to show that $\mathcal{T}_4 \not\subset \mathcal{T}_5$. For that, we give an infinite set of terms schematizable in $\mathcal{T}_5$ but not in $\mathcal{T}_4$. This set is defined by bounded grammars $G = (\mathcal{F}, \mathcal{H}, \mathcal{R}, t^*)$ where $\mathcal{F} = \{+, *, a, b\}$, $\mathcal{H} = \{\widehat{f}\}, t^* = \widehat{f}(n)$ and $\mathcal{R}$ contains the following rules,

$$\begin{aligned}\widehat{f}(0) &\to a \\ \widehat{f}(0) &\to b \\ \widehat{f}(s(n)) &\to \widehat{f}(n) + \widehat{f}(n) \\ \widehat{f}(s(n)) &\to \widehat{f}(n) * \widehat{f}(n)\end{aligned}$$

This grammar schematizes the infinite set $L(G) = \{a, b, a + a, a * a, (b + a) * (a + b), \ldots\}$. It cannot be schematizable in $\mathcal{T}_5$ because we use only two rules to rewrite the terms and to describe this language we use alternately two symbols f and g. These facts allow us to see trivially that primal grammars cannot do that$\lhd$

6.2 B-terms

The class of B-terms is an extension of I-terms [7] but it is a subclass of bounded grammars. We use the same notation as in the ω-terms definition.

Definition 3 The set $\mathcal{T}_6$ of *B-terms* is the smallest set such that

1. $\mathcal{X} \subset \mathcal{T}_6$,
2. if $f \in \mathcal{F}$ and $t_1, \ldots, t_{\alpha(f)} \in \mathcal{T}_6$ then $f(t_1, \ldots, t_{\alpha(f)}) \in \mathcal{T}_6$,
3. if $t \in \mathcal{T}_6$, $p \in \mathcal{P}\mathrm{os}(t)$, $p \neq \epsilon$, $r \in \mathcal{T}_6$ and $N \in \mathcal{C}$ then $\Phi(t[\diamond]_p, N, r) \in \mathcal{T}_6$.
4. if $t_1, t_2 \in \mathcal{T}_6$, $p \in \mathcal{P}\mathrm{os}(t_1)$, $q \in \mathcal{P}\mathrm{os}(t_2)$, $p \neq \epsilon$, $q \neq \epsilon$, $r \in \mathcal{T}_6$ and $N \in \mathcal{C}$ then $\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, N, r) \in \mathcal{T}_6$.

The terms $\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, N, r)$ and $\Phi(t[\diamond]_p, N, r)$ are called *generators*, their semantic is the following

$$\begin{aligned}\Phi(t[\diamond]_p, 0, r) &= r \\ \Phi(t[\diamond]_p, n+1, l) &= t[\Phi(t[\diamond]_p, n, r)]_p \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, 0, r) &= r \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n+1, l) &= t_1[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n, l)]_p \qquad or \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n+1, l) &= t_2[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n, l)]_q\end{aligned}$$

For the comparison with others classes see figure 4.

7 B-Matching

To use B-terms in the field of rewriting as in [6], we need to match a term with a B-term. Thus we give for generators the following semantic

$$\begin{aligned}\Phi(t[\diamond]_p, 0, r) &= r \\ \Phi(t[\diamond]_p, n+1, l) &= t\sigma[\Phi(t[\diamond]_p, n, r)]_p \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, 0, r) &= r \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n+1, l) &= t_1\sigma[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n, l)]_p \qquad or \\ \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n+1, l) &= t_2\sigma[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, n, l)]_q\end{aligned}$$

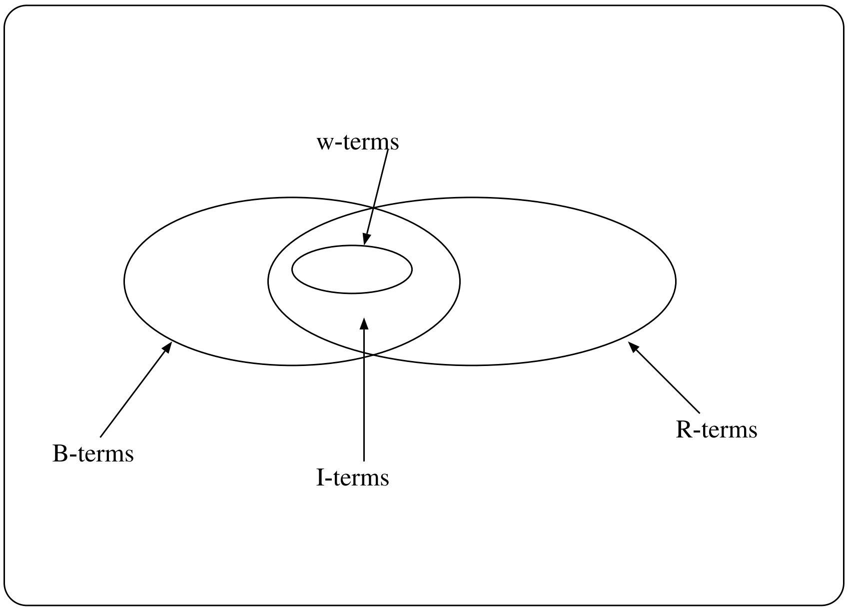

Fig. 4. B-terms, ω-terms, I-terms and R-terms.

where σ is a renaming substitution. Here we give a matching algorithm, called *B-matching*.

We consider two kinds of *equations*. Ordinary equations of the form $s \subseteq t$ where s is a term and t is a B-term and linear diophantine equations over the set of variables $\mathcal{C}$.

A *B-matching problem* is a disjunction of formulae of the form $\exists \vec{N}: \phi \wedge \psi$ where $\vec{N}$ is a vector of integer variables (elements of $\mathcal{C}$), ψ is $\top$, $\perp$ or a conjunction of ordinary equations and ϕ is a disjunction of diophantine equations.

Our goal is to describe the set of transition rules which transform a matching problem ψ to an equivalent problem ψ' such that the two problems have the same set of *solutions*. The transformations continue until obtaining a *solved form*.

Definition 4 A *solved form* is a finite disjunction of formulae of the form

$$\exists \vec{N} . N_1 = E_1 \wedge \ldots \wedge N_k = E_k \wedge t_1 \subseteq x_1 \wedge \ldots \wedge t_n \subseteq x_n$$

where $N_1, \ldots, N_k$ are integer variables which occur only once in the conjunction; $E_1, \ldots, E_k$ are linear expressions over $\vec{N}$; $x_1, \ldots, x_n$ are ordinary variables which occur only once in the conjunction; and $t_1, \ldots t_n$ are B-terms.

Definition 5 A *solution* of a matching problem $s \subseteq t$ is a pair $(\sigma_1; \sigma_2)$ where σ_1 is an unfolder and σ_2 is an ordinary substitution such that $s = t\sigma_1\sigma_2$.

See figure 5 for the transition rules.

Theorem 4 The Matching problem between terms and B-terms is decidable.

Proof: See [1]$\triangleleft$

Example 5 If we have the problem $f(g(a)) \subseteq \Phi(f(\diamond) \mid g(\diamond), N, a)$ then the steps for B-matching are

$$
\begin{array}{ll}
Delete: & \dfrac{t = t}{\top} \\[2ex]
Replace: & \dfrac{(t \subseteq x) \wedge P}{(t \subseteq x) \wedge P\{x \mapsto t\}} \\[2ex]
Decompose: & \dfrac{f(s_1, \ldots, s_n) \subseteq f(t_1, \ldots, t_n)}{(s_1 \subseteq t_1) \wedge \ldots \wedge (s_n \subseteq t_n)} \\[2ex]
Clash: & \dfrac{f(s_1, \ldots, s_n) \subseteq g(t_1, \ldots, t_m)}{\perp} \quad \text{If } f \neq g \\[2ex]
Unfold: & \dfrac{s \subseteq \Phi(t[\diamond]_p, N, r)}{(N = 0 \wedge s \subseteq r) \vee (\exists\, N_1 . N = N_1 + 1 \wedge s \subseteq t\sigma[\Phi(t[\diamond]_p, N_1, r)]_p)} \\[2ex]
UnfoldB: & \dfrac{s \subseteq \Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, N, r)}{\begin{array}{c}(N = 0 \wedge s \subset r) \vee (\exists\, N_1 . N = N_1 + 1 \wedge s \subseteq t_1\sigma[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, N_1, r)]_p) \\ \vee(\exists\, N_2 . N = N_2 + 1 \wedge s \subseteq t_2\sigma[\Phi(t_1[\diamond]_p \mid t_2[\diamond]_q, N_2, r)]_q)\end{array}}
\end{array}
$$

Fig. 5. The Transition Rules for B-Matching a term with a B-term.

$$
\begin{aligned}
f(g(a)) \subseteq \Phi(f(\diamond) \mid g(\diamond), N, a) &\to N = 0 \wedge f(g(a)) \subseteq a \\
&\quad \vee \exists N_1 . N = N_1 + 1 \wedge f(g(a)) \subseteq f(\Phi(f(\diamond) \mid g(\diamond), N_1, a)) \\
&\quad \vee \exists N_2 . N = N_2 + 1 \wedge f(g(a)) \subseteq g(\Phi(f(\diamond) \mid g(\diamond), N_2, a)) \\
&\to (\exists N_1 . N = N_1 + 1) \wedge (g(a) \subseteq \Phi(f(\diamond) \mid g(\diamond), N_1, a)) \\
&\to (\exists N_1 . N = N_1 + 1) \wedge [(N_1 = 0 \wedge g(a) \subseteq a) \\
&\quad \vee \exists N_3 . N_1 = N_3 + 1 \wedge g(a) \subseteq f(\Phi(f(\diamond) \mid g(\diamond), N_3, a)) \\
&\quad \vee \exists N_4 . N_1 = N_4 + 1 \wedge g(a) \subseteq g(\Phi(f(\diamond) \mid g(\diamond), N_4, a)))] \\
&\to (\exists N_1 . N = N_1 + 1) \wedge (\exists N_4 . N_1 = N_4 + 1) \wedge (a \subseteq \Phi(f(\diamond) \mid g(\diamond), N_4, a)) \\
&\to (\exists N_1 . N = N_1 + 1) \wedge (\exists N_4 . N_1 = N_4 + 1) \wedge [(N_4 = 0 \wedge a \subseteq a) \\
&\quad \vee \exists N_5 . N_4 = N_5 + 1 \wedge a \subseteq f(\Phi(f(\diamond) \mid g(\diamond), N_5, a)) \\
&\quad \vee \exists N_6 . N_4 = N_6 + 1 \wedge a \subseteq g(\Phi(f(\diamond) \mid g(\diamond), N_6, a))] \\
&\to (\exists N_1 . N = N_1 + 1) \wedge (\exists N_4 . N_1 = N_4 + 1) \wedge (N_4 = 0 \wedge \top)
\end{aligned}
$$

Then the solution is $(\{N \mapsto 2\}, \{\})$. In this case, the renaming substitution σ is the identity.

8 Automatic Construction of B-terms

We give an algorithm which can be viewed as an improvement algorithm of inductive generalization [12], on terms and as an automatic method of generating B-terms from a sequence of terms. See figure 6.

9 Applications of B-terms

We have seen that matching a term with a B-term is decidable. Thus, the application of B-terms in divergent rewriting systems is possible. We can use B-terms to improve inductive generalization of two terms. For example the classical generalization algorithm[11] of two terms $f(a)$ and $g(a)$ gives a variable x. If we need a precision over the generalization between these terms we can use B-terms and give the generalization $\Phi(f(\diamond) \mid g(\diamond), N, a)$.

We can use B-terms in Logic Programming, too. But we need a unification algorithm between B-terms. We conjecture that unification between B-terms is decidable.

Decompose:

$$\frac{\lambda(f(t_1, \ldots, t_n), f(s_1, \ldots, s_n))}{f(\lambda(t_1, s_1), \ldots, \lambda(t_n, s_n))}$$

Distinct:

$$\frac{\lambda(\underbrace{t[t[\ldots t[}_{n} g(\overrightarrow{s}) \underbrace{]_u]_u \ldots]_u}_{n}, g(\overrightarrow{s'}))}{\Phi(t[\diamond]_u, N, \lambda(g(\overrightarrow{s}), g(\overrightarrow{s'})))}$$

If $(f(\overrightarrow{t})[g(\overrightarrow{s})]_u \not\subseteq g(\overrightarrow{s'})) \wedge (g(\overrightarrow{s'}) \not\subseteq f(\overrightarrow{t})[g(\overrightarrow{s})]_u) \wedge (t(\epsilon) = f)$
and where n is the lest integer such that the first argument of
λ can be decomposed like that and N is a new integer variable

General:

$$\frac{\lambda(s, t)}{\phi(s, t)}$$

If Decompose and Distinct are not applicable

Fig. 6. The Transition Rules for Automatically Generating B-terms from Terms.

References

1. Ali Amaniss. Méthodes de Schématisationon pour la Démonstration Automatique. Doctorat d'Université de Nancy I, Henri Poincaré, in preparation, 1996.
2. Ali Amaniss, Miki Hermann, and Denis Lugiez. Etude comparative des méthodes de schématisation de séquences infinies de termes du premier ordre. Research report 93-R-114, Centre de Recherche en Informatique de Nancy, Nancy, France, 1993.
3. A. Arnold and M. Dauchet. Transductions de Forêts Reconnaissables Monadiques. Forêts Corégulières. *RAIRO Informatique Théorique et applications*, 10(3):5–28, 1976.
4. H. Chen and J. Hsiang. Logic programming with recurrence domains. In J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, editors, *Proceedings 18th ICALP Conference, Madrid (Spain)*, volume 510 of *Lecture Notes in Computer Science*, pages 20–34. Springer-Verlag, July 1991.
5. H. Chen and J. Hsiang. Recurrence domains: their unification and application to logic programming. *Information and Computation*, 122:45–69, 1995.
6. H. Chen, J. Hsiang, and H. C. Kong. On finite representations of infinite sequences of terms. In S. Kaplan and M. Okada, editors, *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, volume 516 of *Lecture Notes in Computer Science*, pages 100–114. Springer-Verlag, June 1990.
7. H. Comon. On Unification of Terms with Integer Exponents. *Mathematical Systems Theory*, 28(1):67–88, 1995.
8. N. Heintze. *Set based program analysis.* PhD thesis, Carnegie Mellon University, 1992.
9. M. Hermann. On the relation between primitive recursion, schematization and divergence. In Hélène Kirchner and G. Levi, editors, *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 115–127. Springer-Verlag, September 1992.
10. Hélène Kirchner. Schematization of infinite sets of rewrite rules generated by divergent completion processes. *Theoretical Computer Science*, 67(2-3):303–332, October 1989.
11. Jean-Louis Lassez, M. J. Maher, and K. Marriot. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan-Kaufman, 1988.
12. Gordon D. Plotkin. A Further Note on Inductive Generalization. *Machine Intelligence*, 6:101–124, 1971.
13. G. Salzer. The unification of infinite sets of terms and its applications. In A. Voronkov, editor, *Proceedings of the 1st International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 409–420. Springer-Verlag, July 1992.