Set Operations for Recurrent Term Schematizations

Ali Amaniss, Miki Hermann, Denis Lugiez

CRIN-INRIA & LEIBNIZ-IMAG [*]

Abstract. Reasoning on programs and automated deduction often require the manipulation of infinite sets of objects. Many formalisms have been proposed to handle such sets. Here we deal with the formalism of recurrent terms proposed by Chen and Hsiang and subsequently refined by several authors. These terms contains iterated parts and counter variables to control the iteration, providing an important gain in expressive power. However, little work has been devoted to the study of these terms as a mechanism to represent sets of terms equipped with the corresponding operations union, intersection, inclusion, membership. In this paper, we focus on the set operations relevant for this schematization formalism and we discuss several possible definitions of these operations. We show how intersection, membership and inclusion can be solved by previously known algorithms and we prove the decidability of the generalisation of two iterated terms, which is the analogy of set union. Moreover, we refine this procedure for computing the generalisation of usual first-order terms using iterated terms, therefore improving Plotkin's algorithm.

1 Introduction

The representation and manipulation of infinite sets of objects constitutes a key problem in automated deduction and logic programming. In fact, theoretical results often imply the existence of an infinite structure (usually a set) but the existing tools, e.g. in programming languages, require the manipulated structures to be finite. Several solutions have been proposed to overcome this problem. One of the simplest solutions consists of using terms with variables that range over the Herbrand universe generated by a given signature. Unfortunately, very often this representation is not expressive enough or does not meet other structural requirements inherent to the schematised set. Other formalisms, like regular tree languages, are easy to manipulate, e.g. using a corresponding tree automaton, since the set operations are easy to realize, but once more they often lack expressive power, since the sets to model are usually not regular. For this purpose, several authors [CH95,HG97,Sal92,Com95] introduced the recurrent schematisations of infinite sets of terms with structural similarities. In these schematisations, the structural similarities are materialised through iterated contexts,

[*] current address: CRIN-INRIA BP 239, 54506 Vandoeuvre les Nancy Cedex FRANCE e-mail: amaniss,hermann,lugiez@loria.fr

where the iteration in a term is controlled by the position and the level of the context. The iteration level is usually expressed by integer variables.

Schematisation formalisms are useful in several branches of logic and automated deduction. They can represent infinite complete sets of unifiers for an equational theory, successive approximations of an infinite or rational tree, an infinite set of answers as a result of an unsafe Datalog query, etc. Such recurrent formalisms can be extremely helpful when we need to reason on program behaviour since we must reason on an infinite set of states bearing some similarity. If each program state is represented by a term then the program development between two states is characterised through the unfolding of a context. Henceforth, the properties usually expressed in temporal logic can be converted to and proved in the formalism of a recurrent schematisation. Another possible application of recurrent schematisations in logic is model construction as explained in [CP96]. Yet another application is the recently developed theory of set constraints [AKW95]. Recurrent schematisations itself can be viewed as a new type of set constraints, where the constraints on terms are expressed by iterations of contexts.

Many of the previously evoked applications of recurrent schematisations require the existence of the set operations, like membership, intersection, inclusion, union, and complement. It is surprising to see that most of the work done on recurrent schematisations deals mainly with matching i.e., membership, and unification i.e., intersection, but there is almost no work done concerning other set operations, apart from the general result on equational problems in the first-order theory of a schematisation called iterated terms [Pel96]. In this work we study the positive set operations on the infinite sets schematised by iterated terms. These operations are membership, intersection, inclusion, and generalisation which is, in some sense, the analogy to union. We discuss several possible definitions for set operations and exhibit examples of properties that are true for first-order terms but false for iterated terms. Another contribution of this paper is a generalisation algorithm which computes an iterated term subsuming two given iterated terms. This specialised algorithm provides a more subtle generalisation of first-order terms which can be especially valuable for applications to model construction. The underlying idea is that two incomparable terms are generalised not to a variable but to a schematised set including the two terms.

2 Definitions

For the sake of simplicity, we have chosen Comon's formalism for iterated terms instead of the more general formalism due to Salzer. Our results could be easily extended to the later framework but the extra complexity of proofs would make the main ideas less clear. Our definitions for iterated terms and their semantics are slightly different from the definitions of [Com95], but the basic idea is the same. Let Σ be a finite set of function symbols where each symbol has a given arity, $\mathcal{X}$ be a denumerable set of first-order variables, $\mathcal{N}$ a denumerable set of integer variables. The set of usual first-order terms is denoted by $T_\Sigma(\mathcal{X})$, and the set of ground first-order terms is denoted by T_Σ.

Definition 1. The class $T_\Sigma(\mathcal{X},\mathcal{N})$ of iterated terms is the smallest set such that

- if $x \in \mathcal{X}$ then $x \in T_\Sigma(\mathcal{X},\mathcal{N})$
- if $s_1,\ldots,s_n \in T_\Sigma(\mathcal{X},\mathcal{N})$, $f \in \Sigma$ and arity of f is n then $f(s_1,\ldots,s_n) \in T_\Sigma(\mathcal{X},\mathcal{N})$,
- if $s,t \in T_\Sigma(\mathcal{X},\mathcal{N}), p \in IPos(t), p \neq \epsilon$ then $t[]_p^N.s \in T_\Sigma(\mathcal{X},\mathcal{N})$

where $IPos(t)$ is the set of iteration positions of t defined by the equalities

- $IPos(x) = \{\epsilon\}$,
- $IPos(f(s,\ldots,s_n)) = \{\epsilon\} \cup_{1\leq i\leq n} i.IPos(s_i)$
- $IPos(t[]_p^N.u) = \emptyset$

Example 2. $s = f((f(\diamond,x))[]_1^N.y,(f(x,\diamond))[]_2^N.z)$ is an iterated term.

In the following, the notation $t[p \leftarrow u]$ denotes the replacement of the subterm of t at position p by the term u, the symbol of t at position p is denoted by $t(p)$, $p \parallel q$ means that neither p is a prefix of q nor the converse. In a term $t[]_p^N.u$, the subterm at position p doesn't really matter and can be safely replaced by a new constant $\diamond$ representing the context hole. From now on, we assume that this replacement is done in each iteration. Iterated terms contains integer variables and first-order variables.

Definition 3. The set $\mathcal{X}$-$Var(u)$ of first-order variables of u is defined by the equalities.

- $\mathcal{X}\text{-}Var(x) = \{x\}$,
- $\mathcal{X}\text{-}Var(f(s_1,\ldots,s_n)) = \cup_{1\leq i\leq n}\mathcal{X}\text{-}Var(s_i)$,
- $\mathcal{X}\text{-}Var(t[]_p^N.s) = \mathcal{X}\text{-}Var(s) \cup \{x \mid \exists q \in Pos(t)\ p \parallel q, t(q) = x\}$

The set $\mathcal{N}$-$Var(u)$ of integer variables of u is defined by the equalities.

- $\mathcal{N}\text{-}Var(x) = \emptyset$
- $\mathcal{N}\text{-}Var(f(s_1,\ldots,s_n)) = \cup_{1\leq i\leq n}\mathcal{N}\text{-}Var(s_i)$,
- $\mathcal{N}\text{-}Var(t[]_p^N.s) = \{N\} \cup \mathcal{N}\text{-}Var(t) \cup \mathcal{N}\text{-}Var(s)$.

A term t s.t. $\mathcal{N}\text{-}Var(t)=\mathcal{X}\text{-}Var(t)=\emptyset$ is called a *ground* term. Substitutions instantiate variables. Since we have two kinds of variables, we have two kinds of substitutions. $\mathcal{X}$-substitutions replace first-order variables by iterated terms and $\mathcal{N}$-substitutions replace integer variables by linear forms.

Definition 4. A $\mathcal{X}$-substitution σ is a finite set of pairs $\{x_1 \leftarrow t_1,\ldots,x_p \leftarrow t_p\}$ where the x_i's belong to $\mathcal{X}$ and the t_i's to $T_\Sigma(\mathcal{X},\mathcal{N})$. The domain of σ, denoted by $Dom(\sigma)$ is the set $\{x_1,\ldots,x_p\}$. The application of σ to a term t is defined by the equalities.

- $x\sigma = t_i$ if x is some x_i, otherwise $x\sigma = x$,
- $f(s_1,\ldots,s_n)\sigma = f(s_1\sigma,\ldots,s_n\sigma)$
- $(t[]_p^N.s)\sigma = t\sigma[]_p^N.(s\sigma)$

The substitution is ground if the t_i's are ground. The substitution such that $Dom(\sigma) = \emptyset$ is denoted by $id_{\mathcal{X}}$.

We define now the unfolding of integer exponents that we need for defining the substitution of integer variables.

Definition 5. The unfolding of a term $t[]_p^K.u$ is defined by the equalities.

- $t[]_p^n.u = \underbrace{t[\ldots t}_{n}[u]\underbrace{_p \ldots]_p}_{n}$,
- $t[]_p^{n.M}.u = (\underbrace{t[\ldots t}_{n}[\diamond]\underbrace{_p \ldots]_p}_{n})[]^M_{\underbrace{p.\ldots p}_{n}}.u$
- $t[]_p^{N+M}.u = t[]_p^M.(t[]_p^M.u)$

Definition 6. A $\mathcal{N}$-substitution σ is a finite set of pairs $\{N_1 \leftarrow \alpha_1, \ldots, N_p \leftarrow \alpha_p\}$ where α_i is a linear form $a_i^0 + \Sigma_{j=1,\ldots,k_i} a_i^j.M_j$ with $a_i^j > 0$ and the M_j are integer variables. The domain of σ is $Dom(\sigma) = \{N_1, \ldots, N_p\}$.The application of σ to a term is defined by the following equalities.

- $x\sigma = x$,
- $f(s_1, \ldots, s_n) = f(s_1\sigma, \ldots, s_n\sigma)$,
- $(t[]_p^N.s)\sigma = t\sigma[]_p^{\alpha_i}.s\sigma$ if N is some N_i, otherwise $(t[]_p^N.s)\sigma = t\sigma[]_p^N.s\sigma$

The substitution is ground when all α_i are positive integers. The substitution such that $Dom(\sigma) = \emptyset$ is denoted by $id_{\mathcal{N}}$.

A $\mathcal{N}, \mathcal{X}$-substitution σ (in short substitution) is a pair (σ_1, σ_2) with σ_1 being a $\mathcal{N}$-substitution and σ_2 being a $\mathcal{X}$-substitution. Substitutions are used to define the semantics of an iterated term, i.e. the set of first-order terms represented via unfolding. Indeed two semantics are possible:

Definition 7. (Semantics of iterated terms)

- $U(s) = \{s\sigma \mid \sigma = (\sigma_1, id_{\mathcal{X}})$ *with* σ_1 *ground* $\mathcal{N}$*-substitutions such that* $\mathcal{N}\text{-}Var(s) \subseteq Dom(\sigma)\}$ (free semantics)
- $UG(s) = \{t\sigma \mid t \in U(s),\ \sigma = (id_{\mathcal{N}}, \sigma_2)$ *with* σ_2 *ground* $\mathcal{X}$*-substitution such that* $\mathcal{X}\text{-}Var(t) \subseteq Dom(\sigma)\}$ (ground semantics)

Example 8. Let $s = f(\diamond, x)[]_1^N.x'$ then $U(s) = \{f(x', x), f(f(x', x), x), \ldots\}$ and $UG(s) = f(T_\Sigma, T_\Sigma)$.

The main difference with previous approaches is **that integer variables can't be assigned the zero value**. That means that each unfolding of a term $(t[]_p^N.s)$ contains at least one occurrence of the pattern $t[]$, therefore all unfoldings have the same root symbol. This is particularly helpful when we consider generalisations since it prevents the association between unrelated terms.

3 Definition issues

When dealing with sets of first-order terms represented by iterated terms, two approaches are possible. The first one deals with the syntactical representations only, for instance unification and matching are typically related to this approach. The second one relies more on the semantics of terms, as for the inclusion operation. In this section we discuss the implications of each aspect.

3.1 Intersection and membership

Intersection and membership can be solved with unification and matching algorithms already developed for iterated terms (for instance see ([Com95] for the description of an unification algorithm). Let s and t be two iterated terms, and let $u_1, \ldots, u_n$ be the most general unifiers of s and t, computed by some unification algorithm. The intersection problem is settled by the next proposition.

Proposition 9. *The statement $v \in U(s) \cap U(t)$ holds iff $v \in U(u_i)$ holds for some i. The statement $v \in UG(s) \cap UG(t)$ holds iff $v \in UG(u_i)$ holds for some i.*

Therefore the most general unifiers $u_1, \ldots, u_n$ can be used to represent the intersection of the terms s and t. Membership is also straightforward.

Proposition 10. *The statement $s \in U(t)$ holds iff $s = t\sigma$ holds for some σ. The statement $s \in UG(t)$ holds iff $s = t\sigma$ holds for some σ.*

The other operations raise more interesting questions.

3.2 Matching and inclusion

Until the end of the section, s and t are two terms which do not share variables. The classical definition for matching is the following.

Definition 11. (Matching) s matches t iff there exists a substitution σ such that $s = t\sigma$ holds.

An immediate corollary is that the inclusions $U(s) \subseteq U(t)$ and $UG(s) \subseteq UG(t)$ hold. For first-order terms the converse is true, i.e., $U(s) \subseteq U(t)$ resp. $UG(s) \subseteq UG(t)$ implies that $s = t\sigma$. Therefore one can ask whether this still holds for iterated terms. The answer is no in both cases.

Example 12. Let us consider the two semantics.

- For the free semantics, let us consider $s = f(f(f(a)))$ and $t = f(f(a))$. Then the terms $s = (f(\diamond))[\ ]_1^N.f(a)$ and the term $t = f((f(\diamond))[\ ]_1^M.a)$ are two iterated terms such that $U(s) = U(t)$ holds but there is no substitution σ with $s = t\sigma$ or $t = s\sigma$.
- For the ground semantics, $s = f(\diamond, x)[\ ]_1^N.x'$ and $t = f(y, \diamond)[\ ]_1^M.y'$. Then $UG(s)$ and $UG(t)$ are both equal to $f(T_\Sigma, T_\Sigma)$ and there is no σ such that $s = t\sigma$ nor the converse hold.

These examples give the grounds for introducing the inclusion predicate:

Definition 13. (Inclusion) We say that s is included in t, written $s \subseteq t$, iff $UG(s) \subseteq UG(t)$ holds.

Inclusion and matching coincide for first-order terms, and it is worthwhile to see if it holds in other cases. Let us ask the following question: if s is a first-order term, t is an iterated term such that $UG(s) \subseteq UG(t)$ holds, does s match t (i.e., $s = t\sigma$ for some σ)? The next example gives the answer.

Example 14. Let $\Sigma = \{a, f\}$ with a constant a and a function symbol f of arity 2. Let $t = f(f(a,a),\diamond)_2^N.f(f(f(z_1,z_2),a),z_3)$ and $s = f(f(a,a), f(f(x,a), f(f(f(a,a),a),a)))$. We have that $s\{x \leftarrow a\} = t\{N \leftarrow 2, z_1 \leftarrow a, z_2 \leftarrow a, z_3 \leftarrow a\}$ and $s\{x \leftarrow f(\alpha,\beta)\} = t\{N \leftarrow 1, z_1 \leftarrow \alpha, z_2 \leftarrow \beta, z_3 \leftarrow f(f(f(a,a),a),a)\}$, therefore $UG(s) \subseteq UG(t)$ holds. On the other hand, $s \neq t\sigma$ for any σ since σ must instantiate the variable N by 1 which is impossible since x clashes with a or by 2 which is forbidden because x clashes with $f(z_1,z_2)$.

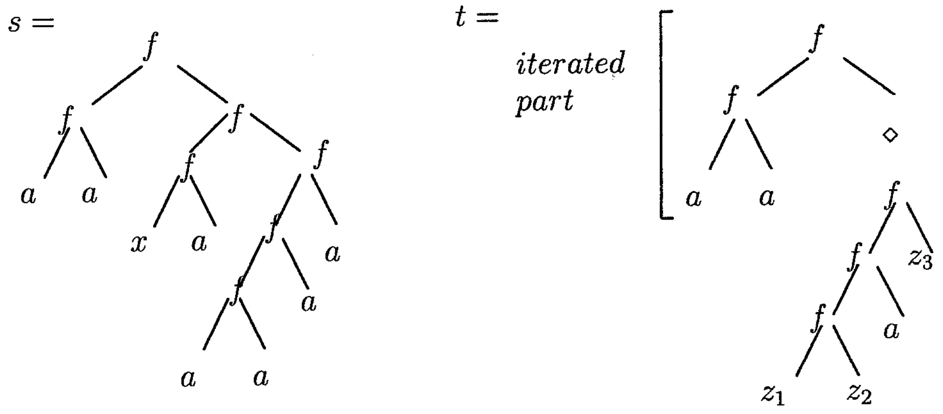

Fig. 1. Two terms such that $UG(s) \subseteq UG(t)$ holds but s doesn't match t

3.3 Generalisation

The same problem occurs for the generalision of iterated terms. Using the same definition as in the first-order case would result in the following one.

Definition 15. Let s and t be two terms, a generalisation of s and t is a term g such that there exist two substitutions σ_1, σ_2 where $g\sigma_1 = s$ and $g\sigma_2 = t$. A generalisation g is minimal if there is no other generalisation g' such that $g' = g\sigma$.

Since a variable is a generalisation of any pair of terms, the most relevant concept is that of a minimal generalisation. The above definition is not really satisfactory, as shown by the following example.

Example 16. Let $s = f(f(f(a)))$ and $t = f(f(a))$, then $g = f(\ f(\diamond)[\ ]_1^N.a\ )$ and $g' = (f(\diamond))[\ ]_1^N.f(a)$ are two generalisations of s and t. It is easy to see that they are minimal but there is no substitution σ such that $g' = g\sigma$ nor $g = g'\sigma$. On the other hand, $U(g) = U(g')$ and $UG(g) = UG(g')$.

Therefore using the same notion of generalisation for iterated terms as for first-order terms leads to counter-intuitive results, since we distinguish between two terms which have the same semantics and that should be therefore identified. This suggests a new definition, where generalisations are compared with respect to the ground semantics:

Definition 17. (ground generalisation) A term g generalises the terms s and t iff there exists two substitutions σ_1, σ_2 such that $s = g\sigma_1$ and $t = g\sigma_2$. The generalisation g is minimal iff there is no other generalisation g' such that $UG(g') \subseteq UG(g)$ where the inclusion is strict.

Another possible definition refers to the meanings of the terms:

Definition 18. (inductive generalisation) A term g generalises the terms s and t iff $UG(s) \subseteq UG(g)$ and $UG(t) \subseteq UG(g)$ hold. The generalisation g is minimal iff there is no other generalisation g' such that $UG(g') \subseteq UG(g)$ holds where the inclusion is strict.

It is straightforward to see that if g generalises s and t according to the ground semantics, it generalises s and t according to inductive generalisation. Therefore the last definition computes more generalisations. This is why we shall use the former definition (ground generalisation) instead of the latter one. However the decidability result that we give holds for both definitions.

4 Inclusion of iterated terms

In this section we indicate how to solve the inclusion problem, i.e. given s, t decide whether $UG(s) \subseteq UG(t)$ holds.

Theorem 19. *The inclusion problem is decidable.*

Proof. Use the general procedure of [Pel96] or [HS96].

However the inclusion problem is a special case of equational formulae and its solution doesn't require the full power of the decision procedure. A simple algorithm has been given by the first author in his thesis [Ama96] when there are no first-order variables in the quantification part, i.e., problems of the form $\forall\mathbf{N}\exists\mathbf{M}\ s = t$ with $\mathbf{N} = \mathcal{N}\text{-}Var(s)$ and $\mathbf{M} = \mathcal{N}\text{-}Var(t)$. The rules are similar to the rules used in the unification algorithm described in [Com95]. Two unfolding rules are used for elimination of quantified variables. Universally quantified variables lead to a conjunction of the base case ($N = 1$) and of the inductive case ($N = 1 + N'$), whereas existential variables lead to disjunction of these cases. Together with the other unification rules, we eventually eliminate the quantifiers through reasoning on unfolding paths in both terms s and t.

5 A brute-force solution for the generalisation problem

First we give some definitions needed by the non-uniqueness of generalisations.

Definition 20. A set of generalisations S for two terms s and t is *complete* iff for each generalisation g of s and t there exists some $g' \in S$ such that the inclusion $UG(g') \subseteq UG(g)$ holds. A set of generalisations is a *complete minimal* set iff it is complete and contains only minimal generalisations.

The following proposition is a straightforward consequence of the definition.

Proposition 21. *A complete minimal set of generalisations is unique modulo the equivalence $s \equiv t$ iff $UG(s) = UG(t)$ holds.*

Now we show that the set $\mu G(s,t)$ of minimal generalisations is finite and algorithmically computable for any terms s and t. The algorithm first computes the finite set of all possible generalisations and eliminates the redundant ones in the second pass using the inclusion decision procedure. The idea behind the algorithm is that instantiation cannot decrease the height of a term. Therefore, a generalisation has a height smaller than or equal to the generalised terms. Since there is only a finite number of terms with a height smaller than a fixed bound, subsequently there are only finitely many generalisations. We are going to state this proof more formally.

Definition 22. The height of a term is defined by:
- $H(x) = 1$, $H(\diamond) = 1$
- $H(f(t_1, \ldots, t_n) = 1 + Max\{H(t_1), \ldots, H(t_n)\}$ for $n \geq 1$,
- $H(t[\,]_p^N.u) = Max\{H(t[p \leftarrow \diamond], |p| + H(u)\}$. where $|p|$ is the length of p.

Example 23. $H(f(f(a,a),\diamond)[\,]_2^N.a) = 3$, $H(f(a,f(a,\diamond))[\,]_{2.2}^N.g(a)) = 4$

Proposition 24. *There are only finitely many terms (up to variable renaming) of height smaller than a given bound n.*

Proposition 25. *Let t be an iterated term and σ a substitution, then the inequality $H(t\sigma) \geq H(t)$ holds.*

Proof. The proof is by structural induction on t.

Proposition 26. *The set of generalisations of two terms s and t is finite.*

Proof. Let g be a generalisation of s and t, then $H(g) \leq H(g\sigma)$, therefore $H(g) \leq min(H(s), H(t))$. There are only finitely many distinct (up to renaming) iterated terms of height smaller than some fixed bound (here $min(H(s), H(t))$, therefore there is only a finite number of possibilities for g.

Theorem 27. *There exists an algorithm to compute a set of minimal generalisations of s and t.*

Proof. Enumerate terms of height smaller than $min(H(s), H(t))$ and check if they generalise s and t. Then use the inclusion algorithm to find minimal ones.

Remark 28. The same property holds for inductive generalisation since the height of any generalisation of s is bounded by $H(s\sigma)$ where σ instantiates the integer variables of s by 1 and the first-order variables of s by a constant.

6 Generalisation of first-order terms using iterated terms

In model construction, a main problem is to describe sets of first-order terms representing a model under construction in a compact way. Several authors have suggested to use iterated terms for such purposes. A weakness of this approach is the generation of iterated terms. The procedure starts with a set of first-order terms and at some point infers a representation of a model containing these terms. The representation must be faithful i.e., must contain the given terms, compact, and not too general. First-order generalisation usually provides us with a result which is too general, for example the generalisation of $f(a)$, $f(f(a))$, $f(f(f(a)))$, ... is $f(x)$ whereas iterated terms can provide us with a better approximation $(f(\diamond))[\;]_1^N.a$. Ad-hoc solutions exist, but there is no systematic treatment of the problem relying on a generalisation algorithm. In this section we provide a generalisation algorithm for first-order terms using iterated terms. For simplicity, we consider only a generalisation of two terms but the algorithm can be easily extended to work on a finite set of terms.

6.1 Is the generalisation of first-order terms unique?

The generalisation of classical terms is unique (up to renaming) and the set of first-order terms equipped with the unification and generalisation operations has a lattice structure. On the other hand unification of iterated terms is finitary and it is likely that generalisation is finitary too. However the unification algorithm for iterated terms applied to first-order terms computes a unique most general unifier. Therefore a natural question is to ask if the same holds for the generalisation of first order terms using iterated terms. The following example shows that this is false whatever definition of generalisation is used.

Example 29. Let the signature be $\Sigma = \{f, h, a, b\}$ with f and h of arity 1 and a, b two constants. Let $s = f(h(f(h(h(a)))))$ and $t = f(h(h(f(b))))$.

Proposition 30. *All definitions of generalisation are equivalent for s and t.*

Since s and t are ground terms $U(s) = UG(s) = \{s\}$ and $U(t) = UG(t) = \{t\}$. Moreover g generalises s and t according to inductive generalisation iff $UG(s) \subseteq UG(g)$ and $UG(t) \subseteq UG(g)$ which yields $s = g\sigma_1$ for some σ_1 and $t = g\sigma_2$ for some σ_2. Therefore g generalises s and t according to ground generalisation.

Proposition 31. *The iterated terms* $\phi_1 = \underbrace{(f(h(\diamond))}_{iterated\ part}\ [\ ]_2^N.h(x)$ *and* $\phi_2 = f(\ \underbrace{h(\diamond)}_{iterated\ part}\ [\ ]_1^N.f(x))$ *are two minimal generalisations of* s *and* t.

Proof. We have $f(g(g(a))) \in UG(\phi_1)$ but $f(g(g(a))) \notin UG(\phi_2)$ and $f(g(f(a))) \in UG(\phi_2)$ but $f(g(f(a))) \notin UG(\phi_1)$. One can check that ϕ_1 and ϕ_2 generalize s and t and that they are minimal.

6.2 Differences from first-order generalisation

The main difficulty of first-order generalisation is illustrated by the following example. Let $s = f(a, a)$ and $t = f(b, b)$ be two terms. A naive algorithm that generalises arguments when the roots are identical and generates new variables for distinct roots would result in $f(x, y)$ when the actual generalisation is $f(x, x)$. The problem is solved by using a bijection between pairs of terms and variables. For iterated terms the problem is more complex.

Example 32. Let $s = f(a)$ and $t = f(f(a))$ be two terms then their first-order generalisation is $f(x)$. Indeed, both terms contain an iteration of $f(\diamond)$ at position 1 with the same term a after the iteration. Therefore a better proposition is $f(\diamond)[\ ]_1^N(a)$, where N is a new variable.

Following the previous example, our generalisation algorithm contains a rule that detects iterations of a common context in the terms s and t to be generalised. In fact, it looks for a path p such that $p \dots p$ (n times) occurs in s and $p \dots p$ (m times) occurs in t. The power but also the additional complexity of our generalisation algorithm resides in this rule. However, a careless generalisation of integer variables causes the same problem as for first-order variables.

Example 33. Let $s = h(f(a), f(b))$ and $t = h(f(f(a)), f(f(b)))$ be two terms. First we can decompose upon h (no possible iteration occurs) and compute the generalisation of $f(a)$ and $f(f(a))$ and the generalisation of $f(b)$ and $f(f(b))$. As seen previously the first one results in $f(\diamond)_1^N.a$ with N a new variable and the second one in $f(\diamond)_1^M.b$ with M a new variable, yielding $h(f(\diamond)_1^N.a, f(\diamond)_1^M.b)$. But the minimal generalisation is $h(f(\diamond)_1^K.a, f(\diamond)_1^K.b)$ (N and M must be identified).

6.3 A generalisation algorithm

The generalisation algorithm described here is intended to be as simple as possible and will be refined later on. The following transformation rules compute a generalisation $G(s, t)$ of two terms s and t, i.e., a term g such that $g\sigma_1 = s$ and $g\sigma_2 = t$. Since the variables of s and t are not instantiated, we assume that s and t are ground. The rules are non-deterministic and using all possible choices, we get a set of generalisations that is denoted by $Gen(s, t)$. Non-minimal generalisations can appear in this set and we use a cleaning rule to get rid of useless generalisation. In the following, $\xi_{\mathcal{X}}$ is a bijection between pairs of terms and $\mathcal{X}$, and $\xi_{\mathcal{N}}$ is a bijection between pairs of integers and $\mathcal{N}$. For any pair of terms, $\mu G(s,t)$ denotes the set of minimal generalisation of s and t.

Rules for generalisation of first-order terms

(Clash)
$G(g(s_1,\ldots,s_m),f(t_1,\ldots,t_n)) \rightarrow \xi_{\mathcal{X}}(g(s_1,\ldots,s_m),f(t_1,\ldots,t_n)) \quad m,n \geq 0$
(Decomposition)
$G(f(s_1,\ldots,s_n),f(t_1,\ldots,t_n)) \rightarrow f(g_1,\ldots,g_n)$ if $g_i \in \mu G(s_i,t_i)$ $1 \leq i \leq n$
(Iteration) $G(s,t) \rightarrow U[\ ]_p^{\xi_{\mathcal{N}}(n,m)}.w$
if there is some position p, two distinct integers n,m greater than 1 such that the following conditions hold:

- for each prefix q of p, the symbols $s(q)$ and $t(q)$ are the same,
- $s = S[\ ]_p^m.v$ and $t = T[\ ]_p^n.u$ where $S = s[p \leftarrow \diamond]$ and $T = t[p \leftarrow \diamond]$,
- $w \in \mu G(u,v)$,
- $U = gen(p, s[p \leftarrow \diamond], t[p \leftarrow \diamond])$ where gen is defined by:
  - $gen(\epsilon, \diamond, \diamond) = \diamond$,
  - $gen(i.q, f(u_1,\ldots,un), f(v_1,\ldots,v_n)) = f(g_1,\ldots,g_{i-1}, gen(q,u_i,v_i), g_{i+1},\ldots,g_n)$ where $g_j \in \mu G(u_i,v_i)$.

Let $Gen(s,t)$ be the set of all terms $G(s,t)$ computable using the three previous rules. This set can contain non-minimal generalisations, as proved by the next example.

Example 34. Applying the last rule to the terms $s = f(a,f(a,g(a)))$ and $t = f(b,g(c))$ at position $p = 2$ and then the decomposition and clash rules, we get $G(s,t) = f(x,\diamond)_2^N.g(y)$ where $x = \xi_{\mathcal{X}}(a,b)$ and $y = \xi_{\mathcal{X}}(c,a)$. On the other hand, applying decomposition first, we find $G(s,t) = f(x,z)$ with $z = \xi_{\mathcal{X}}(f(b,g(a)),g(c))$. The second result is not minimal since the inclusion $UG(f(x,\diamond)_2^N.g(y)) \subseteq UG(f(x,z))$ holds, but the first result is.

Therefore we introduce the cleaning rule:

(Cleaning) $Gen(s,t) \rightarrow Gen(s,t) - \{g\}$ if there is some $g' \in Gen(s,t)$ such that $UG(g') \subseteq UG(g)$ holds.

When the cleaning rule is no longer applicable we set $\mu G(s,t) = Gen(s,t)$.

6.4 Termination, correction and completeness of the algorithm

In this section we set the main properties of the algorithm.

Proposition 35. *The application of the rules* **Clash, Decomposition, Iteration, Cleaning** *terminates.*

Proof. Computing $\mu G(s,t)$ needs to compute $\mu G(u,v)$ for smaller terms only and only a finite number of paths must be considered by the iteration rule.

Proposition 36. *$\mu G(s,t)$ is a complete set of minimal generalisations.*

7 Conclusion

We have described the set operations inclusion and union for the recurrent schematization by iterated terms. We showed that together with the membership and intersection, solved by matching and unification respectively, these set operations can be algorithmically solved within the considered formalism. The inclusion is presented as an extension of matching to infinite schematized sets with its proper semantics. The union operation is based on the generalisation problem, where we applied the new idea that two incomparable terms are generalised by an infinite schematized set containing the given two terms. This improves the usual notion of generalisation due to Plotkin, where incomparable terms were generalised by a variable. We gave a new generalisation algorithm, based on this new idea, that can be applied to several interesting problems in logic programming, knowledge representation, and automated deduction.

Several interesting questions concerning set operations for recurrent schematisations remain to be studied. In particular, it would be interesting to know how can these notions be developed for other existing recurrent formalisms. On the other hand, almost nothing is known concerning the complexity of the decision problem involving the considered set operations, nor about the asymptotic complexity of the existing algorithms. These questions are interesting also in the scope of set constraints by recurrent schematisations, since already the decision problem in the usual set constraint formalism has a high complexity [BGW93]. This complexity classification would allow us to decide upon the practical applicability of the existing formalism.

References

[AKW95] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. *Inf. and Comp.*, 122(1):30–44, 1995.

[Ama96] A. Amaniss. *Méthodes de schématisation pour la démonstration automatique.* PhD thesis, Université Henri Poincaré – Nancy 1, 1996.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings 8th LICS, Montreal (Quebec, Canada)*, 1993.

[CH95] H. Chen and J. Hsiang. Recurrence domains: their unification and application to logic programming. *Inf. and Comp.*, 122:45–69, 1995.

[Com95] H. Comon. On unification of terms with integer exponent. *Mathematical System Theory*, 28(1):67–88, 1995.

[CP96] R. Caferra and N. Peltier. A significant extension of logic programming by adapting model building rules. In *Proceedings ELP'96*, volume 1050 of *LNAI*, pages 51–65, 1996.

[HG97] M. Hermann and R. Galbavý. Unification of infinite sets of terms schematized by primal grammars. *Theoretical Computer Science*, April 1997. to appear.

[HS96] M. Hermann and G. Salzer. Solution of equational problems in the first-order theory of recurrent schematizations. Unpublished manuscript, October 1996.

[Pel96] N. Peltier. Increasing model building capabilities by constraints solving on terms with integer exponents. submitted to J. Symb. Comp., 1996.

[Sal92] G. Salzer. The unification of infinite sets of terms and its applications. In A. Voronkov, editor, *Proceedings of the 1st LPAR, St. Petersburg (Russia)*, volume 624 of *LNAI*, pages 409–420. Springer-Verlag, July 1992.