Weakly Regular Relations and Applications

Sébastien Limet*, Pierre Réty*, Helmut Seidl+

* LIFO - Université d'Orléans, France. {limet,rety}@lifo.univ-orleans.fr

+ Dept. of Computer Science - University of Trier, Germany. seidl@psi.uni-trier.de

Keywords : tree-tuple languages, equational unification, rewriting.

Abstract. A new class of tree-tuple languages is introduced : the weakly regular relations. It is an extension of the regular case (regular relations) and a restriction of tree-tuple synchronized languages, that has all usual nice properties, except closure under complement. Two applications are presented : to unification modulo a rewrite system, and to one-step rewriting.

1 Introduction

Several classes of tree-tuple languages (also viewed as tree relations), have been defined by means of automata or grammars. In particular, a simple one, the *Regular Relations (RR)*, consists in defining regularity as being the one of the tree language (over the product-alphabet) obtained by overlapping the tuple components.

A more sophisticated class, the *Tree-Tuple Synchronized Languages (TTSL)*, is obtained by extending RRs thanks to synchronization constraints between independent branches. TTSLs have first been introduced by means of *Tree-Tuple Synchronized Grammars (TTSG)* and have been applied to equational unification [7], to logic program validation [10], and to one-step rewriting theory [8]. They have next been reformulated in a simpler way by means of *Constraint Systems (CS)*, and applied to rewriting again and to concurrency [4].

RRs have all usual nice properties[1], but expressiveness is poor. It is just the opposite for TTSLs. In particular, CSs are not closed under intersection[2], nor under complement.

In this paper we define an intermediate class, called *Weakly Regular Relations (WRR)*, between RRs and synchronized languages, that has all RRs properties, except closure under complement. Thanks to its expressiveness greater than RRs, WRRs enable to prove new decidability results :

- on unification modulo a rewrite system. Unlike [7], a non-linear goal is allowed. This result is obtained by using the general method of [11] for deciding unifiability with the help of a tree-tuple language.
- on the existential one-step rewriting theory. Unlike [8], non-linear rewrite rules are allowed.

All missing proofs and details can be found in [9]

[1] Except closure under iteration (transitive closure).

[2] Horizontal TTSGs are closed under intersection, provided a more complicated control is used [8], which makes great difficulties. In particular, we do not know if they are then still closed under projection.

2 Regular Relations and Synchronized Languages

Let Σ be an alphabet, i.e. a set of symbols with fixed arities. T_Σ denotes the set of ground terms over Σ. For a position p in $t \in T_\Sigma$, $t|_p$ denotes the subterm of t at position p, and $t(p)$ denotes the symbol occurring in t at position p.

Given a language S of l-tuples, and T of n-tuples, and for $i \in \{1,\ldots,l\}$, $j \in \{1,\ldots,n\}$ the *i,j-join* of S and T is a $l+n-1$-tuple language, denoted $S \bowtie_{i,j} T$, and defined by:

$$\{(s_1,\ldots,s_l,t_1,\ldots,t_{j-1},t_{j+1},\ldots,t_n) \mid (s_1,\ldots,s_l) \in S \wedge (t_1,\ldots,t_n) \in T \wedge s_i = t_j\}$$

$S \bowtie T$ stands for $S \bowtie_{l,1} T$. For $i_1,\ldots,i_k \in \{1,\ldots,l\}$, the *projection* of S on components $i_1,\ldots,i_k$ is the k-tuple language, denoted $\Pi_{i_1,\ldots,i_k}(S)$, defined by:

$$\Pi_{i_1,\ldots,i_k}(S) = \{(s_{i_1},\ldots,s_{i_k}) \mid \forall j \neq i_1,\ldots,i_k, \exists s_j,\ (s_1,\ldots,s_l) \in S\}$$

For tuples $s \in S$, $t \in T$, $s\,t$ denotes the $l+n$-tuple obtained by *concatenation* of s and t.

2.1 Regular Relations

We define regularity for n-ary relations as in [2], i.e. a relation R is regular iff the tree language over the product-alphabet obtained by overlapping the tuple-components is regular.

Formally, let $\Sigma_\oplus$ be the product alphabet defined by $\Sigma_\oplus = (\Sigma \cup \{\perp\}) \times (\Sigma \cup \{\perp\}) - \{\perp\perp\}$ where $\perp$ is a new constant. For $s, t \in T_\Sigma$ we recursively define $s \oplus t \in T_{\Sigma_\oplus}$ by:

$$f(s_1,\ldots,s_n) \oplus g(t_1,\ldots,t_m) = \begin{cases} fg(s_1 \oplus t_1,\ldots,s_n \oplus t_n, \perp \oplus t_{n+1},\ldots,\perp \oplus t_m) \\ \qquad \text{if } n < m \\ fg(s_1 \oplus t_1,\ldots,s_m \oplus t_m, s_{m+1} \oplus \perp,\ldots,s_n \oplus \perp) \\ \qquad \text{otherwise} \end{cases}$$

For instance $f(a,g(b)) \oplus f(f(a,a),b)$ is $ff(af(\perp a,\perp a), gb(b\perp))$. This definition trivially extends to product of k terms $t_1 \oplus \ldots \oplus t_k$.

A n-ary relation $R \subseteq T_\Sigma^n$ is regular iff the tree language $\{t_1 \oplus \ldots \oplus t_n \mid (t_1,\ldots,t_n) \in R\}$ is regular. *RR* stands for *Regular Relation.* For example, $\{(t,t) \mid t \in T_\Sigma\}$ and for given symbols $f, g \in \Sigma$ of same arity, $\{(t, t[f \leftarrow g]) \mid t \in T_\Sigma\}$ are RRs. On the other hand $\{(t, t_{sym}) \mid t \in T_\Sigma,\ t_{sym}$ is the symmetric tree of $t\}$ is not a RR if Σ is not monadic.

2.2 Constraint Systems for Synchronized Languages

Synchronized languages refer to the class of languages defined by means of TTSG with bounded synchronizations of [7]. The exact definition is rather technical and will not be given here. The aim of this section is to define a more uniform way to recognize this class of languages. We use constraint systems (CS), and we consider their least fix-point solutions. A CS can be viewed as a grammar for which a bottom-up point of view is adopted. We will sometimes identify non-terminals with the sets they generate.

Example 1. In the signature $\Sigma = \{f^{\backslash 2}, b^{\backslash 0}\}$ let $Id_2 = \{(t,t) \mid t \in T_\Sigma\}$ be the set of pairs of identical terms. Id_2 can be defined by the following CS:

$$\begin{aligned} &Id_2 \supseteq (b,b) \\ &Id_2 \supseteq (f(1_1, 2_1),\ f(1_2, 2_2))\ (Id_2, Id_2) \end{aligned}$$

where $1_1, 2_1, 1_2, 2_2$ abbreviate pairs (for readability). For example 2_1 means $(2,1)$, which denotes the first component of the second argument (the second Id_2). Note that since 1_1 and 1_2 come from the same Id_2, they represent two identical terms, in other words they are linked (*synchronized*), whereas for example 1_1 and 2_1 are independent.

Example 2. Now if we consider the slightly different CS :

$$\begin{aligned} &X_{sym} \supseteq (b,b) \\ &X_{sym} \supseteq (f(1_1, 2_1),\ f(2_2, 1_2))\ (X_{sym}, X_{sym}) \end{aligned}$$

we get the set $L_{sym} = \{(t, t_{sym}) \mid t_{sym}$ is the symmetric tree of $t\}$.

Example 3. In the signature $\Sigma = \{s^{\backslash 1}, b^{\backslash 0}\}$ let $L_{dble} = \{(s^n(b), s^{2n}(b))\}$. It can be defined by the CS :

$$\begin{aligned} &X_{dble} \supseteq (b,b) \\ &X_{dble} \supseteq (s(1_1),\ s(s(1_2)))\ X_{dble} \end{aligned}$$

General formalization

Assume we are given a (universal) index set N for tuple components. For $I \subseteq N$ and any set M, the set of I-tuples $a : I \to M$ is denoted by M^I. Often, we also write $a = (a_i)_{i \in I}$ provided $a(i) = a_i$ which for $I = \{1, \ldots, k\} \subseteq \mathbb{N}$, is also written as $a = (a_1, \ldots, a_k)$.

Different tree tuple languages may refer to tuples of different length, or, to different index sets. Our constraint variables represent tree tuple languages. Consequently, they have to be equipped with the intended index set. Such an assignment is called *classification*. Accordingly, a *classified* set of tuple variables (over N) is a pair $(\mathcal{X}, \rho)$ where $\rho : \mathcal{X} \to 2^N$ assigns to each variable X a subset of indices. This subset is called the *class* of X. For convenience and whenever ρ is understood, we omit ρ and denote the classified set $(\mathcal{X}, \rho)$ by $\mathcal{X}$. The maximal cardinality of the classes in $\mathcal{X}$ is also called the *width* of $\mathcal{X}$. In particular, in example 1, $N = \{1, 2\}$, $\mathcal{X} = \{Id_2\}$, $\rho(Id_2) = \{1, 2\}$, and the width of $\mathcal{X}$ is 2.

A *constraint system* (*CS*) for tree tuple languages consists of a classified set $(\mathcal{X}, \rho)$ of constraint variables, together with a finite set $\mathcal{E}$ of inequations of the form

$$X \supseteq \square(X_1, \ldots, X_k) \tag{1}$$

where $X, X_1, \ldots, X_k \in \mathcal{X}$ and $\square$ is an operator mapping the concatenation of tuples for the variables X_i to tuples for X. More precisely, let

$$J = \{(i, x) \mid 1 \leq i \leq k, x \in \rho(X_i)\} \tag{2}$$

denote the *disjoint union* of the index sets corresponding to the variables X_i (in example 1, $J = \{(1,1), (1,2), (2,1), (2,2)\}$ abbreviated into $\{1_1, 1_2, 2_1, 2_2\}$). Then $\square$ denotes a mapping $T_\Sigma^J \to T_\Sigma^{\rho(X)}$. Each component of this mapping is specified through a tree expression t which may access the components of the argument tuple and apply constructors from signature Σ. Thus, t can be represented as an element of $T_\Sigma(J)$ where $T_\Sigma(J)$ denotes all trees over Σ which additionally may contain nullary symbols from the index set J.

Consider, e.g., the second constraint in example 1. There, the first component of the operator is given by $t = f(1_1, 2_1)$.

The mapping induced by such a tree t then is defined by

$$t(s_j)_{j\in J} = t\{j \mapsto s_j\}_{j\in J}$$

for every $(s_j)_{j\in J} \in T_\Sigma^J$. Accordingly, $\square$ is given by a tuple $\square \in T_\Sigma(J)^{\rho(X)}$.

Let us collect a set of useful special forms of constraint systems. The constraint (1) is called

- *non-copying* iff no index $j \in J$ occurs twice in $\square$;
- *horizontal* iff for any components $\square_x$, $\square_y$ and any positions p, q, $\square_x|_p = (i, x')$, $\square_y|_q = (i, y')$ implies p and q are positions at the same depth.
- *regular* iff each component of $\square$ is a single constructor application of the form $\square_x = a_x((1,x), \ldots, (n,x))$ for each $x \in \rho(X)$ (a_x being of arity n). Note that regularity implies horizontality.

The whole constraint system is called non-copying, (horizontal, regular) iff each constraint in $\mathcal{E}$ is so.

For example, the CSs that define Id_2 and L_{sym} are non-copying, and horizontal. Moreover Id_2 is regular. On the other hand, L_{dble} is not horizontal.

The class of non-copying CSs corresponds to the class of TTSGs of [7], so recognizes synchronized languages. In the rest of the paper, we only consider *non-copying* CSs even it is not explicitly written.

Proposition 1. *[4] The class of non-copying CSs is closed under union, projection, cartesian product. Moreover membership and emptiness are decidable.*

Proposition 2. *RRs are exactly the languages defined by regular CSs.*

Proof. If top symbols of all components of $\square$ have the same arity, i.e. $\forall x, y \in \rho(X)$, $arity(a_x) = arity(a_y)$, then it is obvious since overlapping tuple-components amounts to synchronize identical positions together. Otherwise, a non-regular CS may define a RR, like for example $\{X \supseteq (s(1_1), f(1_2, 2_1))(X, Y),\ X \supseteq (b, b),\ Y \supseteq b\}$. But it can be transformed into a regular CS by changing the index set of Y into $\rho(Y) = \{2\}$. The CS becomes $\{X \supseteq (s(1_1), f(1_2, 2_2))(X, Y),\ X \supseteq (b, b),\ Y \supseteq b\}$, which is regular. However, for simplicity, we always consider in examples that for any X, $\rho(X) = \{1, 2, \ldots\}$.

A variable assignment is a mapping σ assigning to each variable $X \in \mathcal{X}$ a subset of $T_\Sigma^{\rho(X)}$ (i.e. a set of tuples of ground terms). Variable assignment σ satisfies the constraint (1) iff

$$\sigma(X) \supseteq \{\Box(t_1, \ldots, t_k) \mid t_i \in \sigma(X_i)\} \tag{3}$$

Note that the $\Box$-operator is applied to the cartesian product of the argument sets $\sigma(X_i)$. In particular, the tuples inside the argument sets are kept "synchronized" while tuples from different argument sets may be arbitrarily combined.

The variable assignment σ is a *solution* of the constraint system iff it satisfies all constraints in the system. Since, the operator application in (3) is monotonic, even continuous (w.r.t. set inclusion of tuple languages) we conclude that each constraint system has a unique least solution.

3 Study of Intersection of Synchronized Languages

Finding a CS that recognizes the intersection of two synchronized languages is difficult, precisely because of synchronizations. The first example exhibits the first difficulty : deadlocks.

Example 4. Let L_1 and L_2 recognized respectively by the variables X_1 and X_2 of the following constraint systems :

L_1	L_2
$X_1 \supseteq (1_1, g(1_2))Y_1$	$X_2 \supseteq (f(1_1), 1_2)Y_2$
$Y_1 \supseteq (f(1_1), f(1_2))Z_1$	$Y_2 \supseteq (g(1_1), g(1_2))Z_2$
$Z_1 \supseteq (g(b), b)$	$Z_2 \supseteq (b, f(b))$

Clearly $L_1 = L_2 = \{(f(g(b)), g(f(b)))\}$, So $L_1 \cap L_2$ is not empty but in L_1 occurrences 1 (in the first component) and 2.1 (in the second component) are synchronized whereas occurrences 2 and 1.1 are synchronized in L_2. This produces a deadlock when trying to run the two CSs in parallel to recognize $L_1 \cap L_2$.

More generally, it is possible to encode the Post Correspondence Problem by testing emptiness of the intersection of two synchronized languages.

Theorem 1. *Emptiness of the intersection of synchronized languages is undecidable.*

Proof. Let A and C be two alphabets and ϕ, ϕ' be morphisms from A^* to C^*. Let us consider the two synchronized tree languages $L = \{(\alpha, \phi(\alpha)) | \, \alpha \in A^+\}$ and $L' = \{(\alpha, \phi'(\alpha)) | \, \alpha \in A^+\}$. Let us write $\phi(a_i) = c_{i,1} \ldots c_{i,p_i}$ and $\phi'(a_i) = c'_{i,1} \ldots c'_{i,p'_i}$ for each $a_i \in A$. Then L and L' are recognized by the following CSs[3] :

L	L'	
$X \supseteq (a_i(1_1),\, c_{i,1} \ldots c_{i,p_i}(1_2))\, X$	$X' \supseteq (a_i(1_1),\, c'_{i,1} \ldots c'_{i,p'_i}(1_2))\, X'$	$\forall a_i \in A$
$X \supseteq (a_i(\perp),\, c_{i,1} \ldots c_{i,p_i}(\perp))$	$X' \supseteq (a_i(\perp),\, c'_{i,1} \ldots c'_{i,p'_i}(\perp))$	$\forall a_i \in A$

So deciding whether $L \cap L'$ is empty amounts to decide the existence of $\alpha \in A^+$ such that $\phi(\alpha) = \phi'(\alpha)$, i.e. solving the Post correspondence problem.

[3] $\perp$ just marks word ends.

Therefore, since emptiness of synchronized languages is decidable [4], this class is not effectively closed under intersection.

Considering the two previous examples, it seems unavoidable to consider a subclass of synchronized languages to get closure under intersection. The first idea is to avoid deadlocks by forbidding leaning synchronizations (i.e. imposing that synchronization points are always at the same depth) : from now we consider only horizontal CSs.

Example 5. Consider again languages Id_2 and L_{sym} as defined in Examples 1 and 2. So $Id_2 \cap L_{sym}$ is the set of pairs of terms (t, t') such that $t = t'$ and t is the symmetric tree of t'. This means that t is a self-symmetric tree. So $Id_2 \cap L_{sym}$ is the set $\{(b, b)\} \cup \{(f(t, t_{sym}), f(t, t_{sym}))\}$ where t_{sym} denotes the symmetric tree of t. Let $X = \{(t, t_{sym}, t, t_{sym})\}$, $Id_2 \cap L_{sym}$ can be defined by

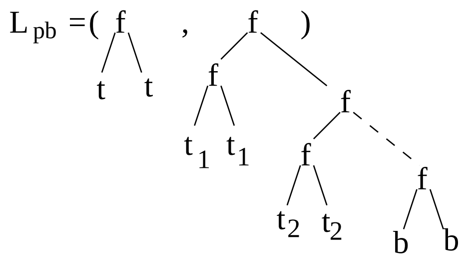
$$\begin{aligned} Id_2 \cap L_{sym} &\supseteq (b, b) | (f(1_1, 1_2), f(1_3, 1_4))X \\ X &\supseteq (f(1_1, 2_1), f(2_2, 1_2), f(1_3, 2_3), f(2_4, 1_4))(X, X) | (b, b, b, b) \end{aligned}$$

It seems that horizontality is a good criterion to get closure under intersection, but unfortunately it is not enough as shown by the following example.

Example 6. Let L_{pb} be the language defined by the following constraints :

$$L_{pb} \supseteq (f(1_1, 1_2), f(2_1, 2_2))(Id_2, L_{pb}) \qquad L_{pb} \supseteq (b, b)$$

The following picture gives an intuition on how L_{pb} looks like :



The intersection of L_{pb} with Id_2 gives the language of pairs of identical balanced terms (i.e. terms all branches of which have the same length). To express it with a non-copying CS, we have to synchronize all occurrences of the same depth together, which requires wider and wider tuples, then infinitely many intermediate languages. This is impossible because a CS is always supposed to be finite. That is why the definition of WRRs needs restrictions stronger than horizontality.

4 Weakly Regular Relations

A horizontal constraint system $\mathcal{C} = (\mathcal{X}, \mathcal{E})$ is *weakly regular* iff there is a rank function $r : \mathcal{X} \to \mathbb{N}$ s.t. for every constraint c given by $X \supseteq \square(X_1, \ldots, X_k)$:

- $r(X) \geq r(X_1) + \ldots + r(X_k)$; and

– $r(X) = r(X_1) + \ldots + r(X_k)$ only provided c is regular.

In essence, this definition implies that each tree tuple of any variable X in the system is constructed by using at most $r(X)$ horizontal but non-regular constraints.

Accordingly, a (set of tree tuples or a) tree relation is called *weakly regular* (*WRR* stands for Weakly Regular Relation) iff it is defined by a weakly regular constraint system.

Example 7. Let $\Sigma = \{f^{\backslash 2}, g^{\backslash 2}, a^{\backslash 0}\}$, consider the rewrite system $R = \{f(x, y) \to g(y, x)\}$, and let $S = \{(t_1, t_2) \mid t_1 \to_R t_2\} = \{C[f(x, y)\sigma], C[g(y, x)\sigma]\}$. S is a WRR:

$$\begin{array}{cc} S \supseteq (f(1_1, 2_1), f(1_2, 2_2))(S, Id_2) & S \supseteq (g(1_1, 2_1), g(1_2, 2_2))(S, Id_2) \\ S \supseteq (f(1_1, 2_1), f(1_2, 2_2))(Id_2, S) & S \supseteq (g(1_1, 2_1), g(1_2, 2_2))(Id_2, S) \end{array}$$
$$S \supseteq (f(1_1, 2_1), g(2_2, 1_2))(Id_2, Id_2)$$
$$Id_2 \supseteq (f(1_1, 2_1), f(1_2, 2_2))(Id_2, Id_2) \quad Id_2 \supseteq (g(1_1, 2_1), g(1_2, 2_2))(Id_2, Id_2)$$

with $r(Id_2) = 0,\ \ r(S) = 1$.

Fact 1 – *If the constraint system $\mathcal{C}$ is weakly regular, then it is weakly regular for a rank function with maximal rank $2^{|\mathcal{C}|}$ (where $|\mathcal{C}|$ denotes the number of variables of $\mathcal{C}$.*
– *It can be decided in linear time whether or not a constraint system is weakly regular.*

There is another structural characterization of weakly regular constraint systems. For variable X, let us denote by $\mathcal{C}_X$ the restriction of the constraint system $\mathcal{C}$ to all variables possibly influencing X. Let us call a constraint $X \supseteq \Box(X_1, \ldots, X_k)$ *recursive* iff some variable X_j on the right-hand side depends on X, i.e., X occurs in $\mathcal{C}_{X_j}$. Then we have:

Theorem 2. *A horizontal constraint system $\mathcal{C}$ is weakly regular iff every recursive constraint $X \supseteq \Box(X_1, \ldots, X_k)$ is regular where furthermore the constraint systems $\mathcal{C}_{X_i}$ are regular for all i except at most one.*

Example 7 defines a WRR indeed, since every recursive rule is regular and Id_2 is a RR. On the other hand, L_{sym} (Example 2) is not a WRR.

Our main new result is:

Theorem 3. *Weakly regular relations are closed under union, projection, cartesian product and intersection.*

The first three closure properties of WRRs stated in theorem 3 are obtained simply by considering the corresponding constructions for arbitrary tree tuple constraint systems and verifying that these preserve the WRR property. In the sequel, we therefore concentrate on the proof of closure under intersection. We need the following auxiliary notion.

A constraint system $\mathcal{C}$ is *single-permuting*, iff every non-regular constraint of $\mathcal{C}$ has only one variable on the right-hand side, i.e., is of the form

$$X \supseteq \square(X_1)$$

The proof of theorem 3 is based onto the following two auxiliary lemmas:

Lemma 1. *Let $\mathcal{C}$ be a weakly regular constraint system. Then an equivalent (up to further auxiliary variables) weakly regular constraint system $\mathcal{C}'$ can be constructed which is single-permuting.*

If $\mathcal{C}$ has maximal rank r, maximal size of classes d, at most a variables in right-hand sides and size n, then $\mathcal{C}'$ has size $\mathcal{O}(n^{(a+1)^r})$ and classes of size $\mathcal{O}(d \cdot a^r)$ where neither the rank nor the number of variables in right-hand sides has increased. Moreover, the constraint system $\mathcal{C}'$ can be constructed in double-exponential time.

Lemma 2. *Assume that the tree-tuple language L of class I is defined by a single-permuting constraint system $\mathcal{C}$ and $\sim \subseteq I \times I$ is an equivalence relation. Then a single-permuting constraint system $\mathcal{C}'$ can be constructed for the language*

$$L^{\sim} = \{t \in L \mid \forall (i,j) \in \sim : t_i = t_j\}$$

In particular, if $\mathcal{C}$ was weakly regular, then so is $\mathcal{C}'$. If $\mathcal{C}$ is of size n and has classes of size at most d, then $\mathcal{C}$ can be constructed in time $d^{\mathcal{O}(d)} \cdot n$.

Using lemmas 1 and 2, the proof of theorem 3 proceeds as follows. Assume we are given languages L_1 and L_2 both defined by weakly regular constraint systems $\mathcal{C}_i$, $i = 1, 2$. By closure under cartesian product, the language $L = \{t^{(1)}t^{(2)} \mid t^{(i)} \in L_i\}$ is defined by a weakly regular constraint system $\mathcal{C}$. By lemma 1, we also replace $\mathcal{C}$ with a single-permuting (weakly regular) constraint system $\mathcal{C}'$. Now consider the equivalence relation $\sim$ on the index set of L which equates the corresponding components from L_1 and L_2. By lemma 2, we can construct from $\mathcal{C}'$ a constraint system $\mathcal{C}''$ which defines $L^{\sim}$. Finally, we may (due to closure under projection) construct a weakly regular constraint system for the language

$$\{t^{(1)} \mid \exists t^{(2)} : t^{(1)}t^{(2)} \in L^{\sim}\} = L_1 \cap L_2$$

and we are done. Calculating the costs of the individual construction steps, we furthermore find that the overall construction can be implemented in double-exponential time.

As an immediate corollary of theorem 3, we obtain:

Corollary 1. *Weakly regular relations are closed under joins.*

Proof. Assume L_1 (resp. L_2) is a l-tuple (resp. n-tuple) language.

$$\begin{aligned} L_1 \bowtie_{i,j} L_2 = {} & \Pi_{1,\ldots,l+j-1,l+j+1,\ldots,l+n}(L_1 \times L_2) \\ & \cap \{(t_1, \ldots, t_i, \ldots, t_l, t_{l+1}, \ldots, t_{l+j-1}, t_i, t_{l+j+1}, \ldots, t_{l+n}) \mid \forall k, t_k \in T_\Sigma\} \end{aligned}$$

Non-Closure under Complement

Consider the balanced full binary trees. Balanced means that all leaves appear at the same depth, and it is well known that a full binary tree t is balanced iff for all non-leaf position v in t, $t|_{v.1} = t|_{v.2}$. Therefore t is unbalanced iff there is a non-leaf position v and a position u s.t. $t(v.1.u) \neq t(v.2.u)$.

Full binary trees are simulated by terms (with fixed arities) over the signature $\Sigma = \{f^{\backslash 2}, a^{\backslash 0}\}$. Thus the set of unbalanced full binary trees is the set of unbalanced terms over Σ, which is generated by variable V of the following constraint system :

$$\begin{array}{ll} V \supseteq f(1_1, 2_1)(V, Id) & U \supseteq (f(1_1, 2_1),\ f(1_2, 2_2))(U, Id^2) \\ V \supseteq f(1_1, 2_1)(Id, V) & U \supseteq (f(1_1, 2_1),\ f(1_2, 2_2))(Id^2, U) \\ V \supseteq f(1_1, 1_2)\,U & U \supseteq (a,\ f(Id, Id)) \\ & U \supseteq (f(Id, Id),\ a) \end{array}$$

where $Id = T_\Sigma$ and $Id^2 = Id \times Id$. It is a WRR since Id is regular and the only non-regular constraint is $V \supseteq f(1_1, 1_2)U$, which is not recursive.

The complement of unbalanced terms, i.e. the balanced ones, cannot be defined by a WRR since, as shown in Section 3, they cannot be defined by a non-copying constraint system.

5 Application to R-unification

This section addresses the problem of unification modulo a confluent constructor-based rewrite system. The goal is to decide the existence of data-unifiers (unifiability), and to express ground data-unifiers by a tree-tuple language.

Under some restrictions, a decidability result has been established using TTSGs [7]. Next the method has been generalized [11] : any class of tree-tuple language can be used, provided :

1. it is closed under join,
2. emptiness is decidable,
3. it can express the tuple-set $N_f = \{(r, \sigma x_1, \ldots, \sigma x_n) \mid f(x_1, \ldots, x_n) \rightsquigarrow^*_{[\sigma]} r$ and $r, \sigma x_1, \ldots, \sigma x_n$ are ground data-terms$\}$ for each defined function f,
4. For each constructor c, it can express the tuple-set

$$N_c = \{(c(t_1, \ldots, t_n), t_1, \ldots, t_n) \mid t_i \text{ are ground data-terms}\}$$

If in addition it is closed under intersection[4] and projection, the goal to be unified may be non-linear.

The unification method presented in [11] has been used with TTSGs and with primal grammars [5] providing some decidable subclasses of R-unification problems. In this section, we present the unification method of [11] which can be used with WRRs getting a new subclass of decidable R-unifiction problems.

[4] This implies the closure under join.

The restrictions are those needed for TTSGs, except that the goal and the non-recursive rewrite rules may be non-linear[5], additional technical restrictions are needed otherwise unifiability is undecidable as shown in [7].

Let us first recall the principle of the general method, using a simple example: *Example 8.*

$$R = \{f(s(x)) \xrightarrow{r_1} p(f(x)), \quad f(p(x)) \xrightarrow{r_2} s(f(x)), \quad f(0) \xrightarrow{r_3} 0\}$$

where $s, p, 0$ are constructors, and consider the linear goal $p(f(x)) \doteq f(s(f(x')))$. We assume that we have a tree-tuple language that satisfies the above properties. In particular it can express N_f, N_s, N_p.

The method consists in simulating the innermost narrowing derivations issued from the goal. We compute:

$$\begin{aligned} N = N_p \bowtie N_f &= \{(s_1, s_2, t) \mid (s_1, s_2) \in N_p \wedge (s_2, t) \in N_f\} \\ &= \{(s_1, s_2, t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(s_2) = s_1\} \end{aligned}$$

$$\begin{aligned} N' &= N_f \bowtie N_s \bowtie N_f \\ &= \{(s'_1, s'_2, s'_3, t') \mid (s'_1, s'_2) \in N_f \wedge (s'_2, s'_3) \in N_s \wedge (s'_3, t') \in N_f\} \\ &= \{(s'_1, s'_2, s'_3, t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(s'_3)) = f(s'_2) \rightarrow^* s'_1\} \end{aligned}$$

From narrowing properties [3], there exists a data-unifier iff there exist instances t, t' such that $s_1 = s'_1$, i.e. such that $N \bowtie_{1,1} N' \neq \emptyset$. Moreover $N \bowtie_{1,1} N'$ expresses the solutions thanks to t, t'.

Now if the goal to be unified is not linear, like $p(f(x)) \doteq f(s(f(x)))$, t must be in addition equal to t'. By projection, we keep only t, t', s_1, s'_1, and force equalities by intersection. Thus, there exists a data-unifier iff $\Pi_{1,3}(N) \cap \Pi_{1,4}(N') \neq \emptyset$.

For each narrowing step occurring within a narrowing derivation $f(x) \rightsquigarrow^*_{[\sigma]} r$ where r is a ground data-term, either s is added in σ and p in r (using r_1), or the opposite (using r_2), and the derivation ends by adding 0 in both (using r_3). So N_f can be described by the expression:

$$N_f \equiv ((p, s) \cup (s, p))^*.(0, 0)$$

On the other hand

$$\begin{aligned} N_s &= \{(s(t), t)\} \equiv (s, \epsilon).((s, s) \cup (p, p))^*.(0, 0) \\ N_p &= \{(p(t), t)\} \equiv (p, \epsilon).((s, s) \cup (p, p))^*.(0, 0) \end{aligned}$$

where ϵ is the empty string. Consequently,

$$\begin{aligned} N &= (p, \epsilon, \epsilon).((s, s, p) \cup (p, p, s))^*.(0, 0, 0) \\ N_f \bowtie N_s &= (p, s, \epsilon).((p, s, s) \cup (s, p, p))^*.(0, 0, 0) \\ N' = (N_f \bowtie N_s) \bowtie N_f &= (p, s, \epsilon, \epsilon).((p, s, s, p) \cup (s, p, p, s))^*.(0, 0, 0, 0) \end{aligned}$$

$$\begin{aligned} \Pi_{1,3}(N) \cap \Pi_{1,4}(N') &= (p, \epsilon).((s, p) \cup (p, s))^*.(0, 0) \bigcap (p, \epsilon).((p, p) \cup (s, s))^*.(0, 0) \\ &= (p(0), 0) \end{aligned}$$

The solutions are given by the second component. So there is one solution: $x/0$.

This is correct: if x/t is a data-unifier of $p(f(x)) \doteq f(s(f(x)))$, necessarily $p(f(t)) = p(f(f(t)))$, then $f(t) = f(f(t))$, then $f(t) = t$ because $f(f(t)) = t$. Therefore $t = 0$.

[5] WRRs are closed under intersection and projection.

Using WRRs

$\boldsymbol{N_c}$ **:** unfortunately, the sets $N_c = \{(c(t_1, \ldots, t_n), t_1, \ldots, t_n)\}$ cannot be expressed by WRRs (nor by RRs), because generating two copies of t_i needs synchronization between them, and one occurs on top and the other at depth 1. So this language is not horizontal.

We slightly modify the method so that N_c is horizontal. Now, using an extra symbol $\natural$, we define $N_c = \{(c(t_1, \ldots, t_n), \natural t_1, \ldots, \natural t_n)\}$ for each constructor c, and $\natural L = \{(\natural t_1, \ldots, \natural t_n) \mid (t_1, ..., t_n) \in L\}$ for any language L. Their constraint systems are:

$$N_c \supseteq (c(1_1, \ldots, n_1), \natural 1_2, \ldots, \natural n_2)(Id_2, \ldots, Id_2)$$
$$\natural L \supseteq (\natural 1_1, \ldots, \natural 1_n)(L)$$

where $Id_2 = \{(t, t) \mid t \in T_C\}$ and T_C is the set of constructor-terms. These constraints are not recursive and Id_2 is regular. So N_c is a WRR and if L is regular, $\natural L$ is a WRR.

It is however necessary to slightly modify the computation of N and N'.

Example 9. Consider the previous example again. Now:

$$\begin{aligned} N = N_p \bowtie \natural N_f &= \{(r_1, \natural r_2, \natural t) \mid (r_1, \natural r_2) \in N_p \wedge (\natural r_2, \natural t) \in \natural N_f\} \\ &= \{(r_1, \natural r_2, \natural t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(r_2) = r_1\} \end{aligned}$$

$$\begin{aligned} N' &= N_f \bowtie N_s \bowtie \natural N_f \\ &= \{(r'_1, r'_2, \natural r'_3, \natural t') \mid (r'_1, r'_2) \in N_f \wedge (r'_2, \natural r'_3) \in N_s \wedge (\natural r'_3, \natural t') \in \natural N_f\} \\ &= \{(r'_1, r'_2, \natural r'_3, \natural t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(r'_3)) = f(r'_2) \rightarrow^* r'_1\} \end{aligned}$$

If the goal is linear, we check $N \bowtie_{1,1} N' \neq \emptyset$ as previously. Otherwise we can still force $t = t'$ by computing $\Pi_{1,3}(N) \cap \Pi_{1,4}(N')$, because the number of $\natural$ above t in N and t' in N' are the same (one). This comes from the fact that the number of constructors appearing above the two occurrences of x in the goal is the same (one).

Definition 1. *For a term t and $u \in Pos(t)$, let $\|u\|$ denote the number of constructors appearing in t above u. The term t is* weak-horizontal *if*

$$\forall u, u' \in Pos(t),\ t(u) = t(u') = x \in Var(t) \implies \|u\| = \|u'\|$$

In this case we also define $\|x\| \stackrel{def}{=} \|u\|$. The goal $t \doteq t'$ (resp. the rewrite rule $l \rightarrow r$) is weak-horizontal *if $t \doteq t'$ (resp. $l \rightarrow r$) is a weak-horizontal term, considering $\doteq$ (resp. $\rightarrow$) is a binary symbol.*

Lemma 3. *If the goal is weak-horizontal and x is a variable occurring several times in the goal, then every component in N, N' that gives the instances of an occurrence of x contains exactly $\|x\|$ times $\natural$.*

Proof. When using n-ary symbols, the intermediate language N^u corresponding to position u ($N^\epsilon = N$) is

$$N^u = (((N_c \bowtie_{2,1} \natural N^{u.1}) \bowtie_{3,1} \natural N^{u.2}) \ldots \bowtie_{n+1,1} \natural N^{u.n}$$

if $t(u)$ is a constructor, and

$$N^u = (((N_f \bowtie_{2,1} N^{u.1}) \bowtie_{3,1} N^{u.2}) \ldots \bowtie_{n+1,1} N^{u.n}$$

otherwise. If x occurs below u, by induction we get that the number of $\natural$ in the component giving the instances of x is exactly the number of constructors occurring along the path from u to x.

Thus, if the goal is weak-horizontal the modified method works, provided N_f can be expressed.

$\boldsymbol{N_f}$ **:** let us explain how to transform a rewrite system into a constraint system that expresses the sets N_f. We give an example before the general algorithm.

Example 10. Consider the rewrite rule $f(c(x,y)) \to d(f(y),x)$. By narrowing, we get $f(x) \leadsto_{[\sigma = x/c(x,y)]} t' = d(f(y),x)$. So $(t',\sigma) = (d(f(y),x),\, c(x,y))$. To get ground data-terms, x should be instantiated by something and $f(y)$ should be narrowed further. Then the corresponding constraint is:

$$F \supseteq (d(1_1, 2_1),\, c(2_2, 1_2))(F, Id_2)$$

Definition 2. *A function position p in a term t is* shallow *if $t|_p = f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are variables.*

Definition 3. *Given a rewrite system R, we define an ordering on defined function symbols by $f > g$ if $f(t_1, \ldots, t_n) \to r \in R$ and g occurs in r. $f \equiv g$ means $f > g \wedge g > f$. The rewrite rule $f(t_1, \ldots, t_n) \to r \in R$ is* recursive *if there is a function g in r s.t. $f \equiv g$.*

Algorithm : we assume that function calls in rhss are shallow, and recursive rewrite rules are linear. Let us write rewrite rules as follows: $f(t_1, \ldots, t_n) \to C[u_1, \ldots, u_m]$ where C is a constructor context containing no variables and each u_i is either a variable or a function call of the form $f_i(x_1^i, \ldots, x_{k_i}^i)$. For each rewrite rule, we create the constraint

$$F \supseteq (C[1_1, \ldots, m_1], \theta t_1, \ldots, \theta t_n)(X_1, \ldots, X_m)$$

where $X_i = Id_2$ if u_i is a variable and $X_i = F_i$ if $u_i = f_i(x_1^i, \ldots, x_{k_i}^i)$ and θ is the substitution $\{u_i/i_2 \mid u_i \text{ is a variable}\}$.

If this rewrite rule is not linear and not recursive, the F_is are defined independently of F. So we can make some variables equal by computing the intersection[6] of the constraint argument $(X_1, \ldots, X_n)$ with the regular relation

$$\{(s_1, \ldots s_k, t, s_{k+2}, \ldots s_p, t, s_{p+2}, \ldots, s_q) | s_i, t \in T_C\}$$

However, the resultant constraint system is not necessarily a WRR: in the previous example, the constraint is recursive and not regular. The non-regularity comes from the fact that when going through the leaves from left-to-right, we get x, y for the lhs, and y, x for the rhs: there is a variable permutation. If we remove the permutation by considering the rule $f(c(x,y)) \to d(x, f(y))$, the resultant constraint is regular. But the absence of variable permutation does not ensure regularity.

[6] if one of the F_is is not a WRR, the algorithm fails.

Example 11. Let $f(a,\, c(x,y)) \to s(f(x,y))$.
The corresponding constraint $F \supseteq (s(1_1), a, c(1_2, 1_3))\, F$ is not regular because there is an internal synchronization in the third component.

And the presence of permutation does not imply non-regularity.

Example 12. Let $f(c(x,y),\, s(z)) \to c(f(x,z),\, y)$.
The corresponding constraint $F \supseteq (c(1_1, 2_1),\, c(1_2, 2_2),\, s(1_3))(F, Id_2)$ is regular.

This is why we introduce the following definition :

Definition 4. *A constructor-based rewrite system is* weak-regular *if function positions in rhs's are shallow, its recursive rules are linear, and the corresponding constraint system generated by the above algorithm is a WRR.*

Note that weak-regularity implies in particular weak-horizontality of rewrite rules. Thanks to decidability of emptiness for WRRs, we get :

Theorem 4. *The unifiability of a weak-horizontal goal modulo a weak-regular confluent constructor-based rewrite system is decidable. Moreover, the unifiers can be expressed by a WRR.*

6 Application to One-Step-Rewriting

Given a signature Σ, the theory of one-step rewriting for a finite rewrite system R is the first order theory over the universe of ground Σ-terms that uses the only predicate symbol $\to$, where $x \to y$ means x rewrites into y by one step.

It has been shown undecidable in [16]. Sharper undecidability results have been obtained for some subclasses of rewrite systems, about the $\exists^*\forall^*$-fragment [15, 12] and the $\exists^*\forall^*\exists^*$-fragment [17].

It has been shown decidable in the case of unary signatures [6], in the case of linear rewrite systems whose left and right members do not share any variables [2][7], for the positive existential fragment [13], for the whole existential fragment in the case of quasi-shallow[8] rewrite systems [1] and also in the case of linear, non-overlapping, non-ϵ-left-right-overlapping[9] rewrite systems [8].

Thanks to WRRs, we get a new result about the existential fragment.

Definition 5. *A rewrite system R is* ϵ-left-right-clashing *if for all rewrite rule $l \to r$, $l(\epsilon) \neq r(\epsilon)$. R is* horizontal *if in each rewrite rule, all occurrences of the same variable appear at the same depth.*

Note that ϵ-left-right-clashing excludes collapsing rules.

Theorem 5. *The existential one-step rewriting theory is decidable in the case of ϵ-left-right-clashing horizontal rewrite systems.*

[7] Even the theory of several-step rewriting is decidable.
[8] All variables in the rewrite rules occur at depth one.
[9] I.e. no left-hand-side overlaps on top with the corresponding right-hand-side.

Since quasi-shallowness is a particular case of horizontality, our result extends that of [1], except that we assume in addition ϵ-left-right-clashing. Compared to [8], rewrite rules may now be non-linear and overlapping, but they must be horizontal.

Consider a finite rewrite system $R = \{ru_1, \ldots, ru_n\}$ and an existential formula in the prenex form. Our decision procedure consists in the following steps:

1. Since the symbols of Σ are not allowed in formulas, every atom is of the form $x \to x$ or $x \to y$. Because of ϵ-left-right-clashing, $x \to x$ has no solutions and is replaced by the predicate without solutions $\perp$, and $x \to y$ is replaced by the equivalent proposition $(x \overset{?}{\to}_{[ru_1]} y \vee \ldots \vee x \overset{?}{\to}_{[ru_n]} y) \wedge x \neq y$, where $\overset{?}{\to}_{[ru_i]}$ is the rewrite relation in zero or one step with rule ru_i. Next, the formula is transformed into a disjunction of conjunctions of (possibly) negations of atoms of the form $x \overset{?}{\to}_{[ru_i]} y$ or $x = y$. We show that the set of solutions of each atom, and of its negation, is a WRR.

2. The solutions of a conjunctive factor are obtained by making cartesian products with the set T_Σ of all ground terms (which is a particular WRR), as well as intersections. For instance let $C = x \overset{?}{\to}_{[ru_1]} y \wedge \neg(y \overset{?}{\to}_{[ru_2]} z)$. The solutions of $x \overset{?}{\to}_{[ru_1]} y$, denoted $SOL(x \overset{?}{\to}_{[ru_1]} y))$, and those of $\neg(y \overset{?}{\to}_{[ru_2]} z)$, denoted $SOL(\neg(y \overset{?}{\to}_{[ru_2]} z))$, are WRRs of pairs, then we can compute $SOL(C) = SOL(x \overset{?}{\to}_{[ru_1]} y)) \times T_\Sigma \ \cap \ T_\Sigma \times SOL(\neg(y \overset{?}{\to}_{[ru_2]} z))$, which is still a WRR (of triples).

3. The validity of the formula is tested by applying the WRR emptiness test on every disjunctive factor.

$SOL(x = y)$ and $SOL(x \neq y)$ are trivially WRRs since they are RRs.

Lemma 4. *$SOL(x \overset{?}{\to}_{[ru_i]} y)$ and $SOL(\neg(x \overset{?}{\to}_{[ru_i]} y))$ are WRRs.*

7 Further work and Conclusion

Computing descendants through a rewrite system may give rise to several applications. [14] shows that the set of descendants of a regular tree language through a rewrite system is still regular, assuming some restrictions. Using non-regular languages, like WRRs, still closed under intersection (for applications), could extend the result of [14] by weakening the restrictions.

Compared to automata with (dis)equality constraints [2], WRRs can define more constraints than only (dis)equality, but they cannot define the balanced terms. However, WRRs and the subclass of reduction automata have something in common: when deriving (recognizing) a term, the number of non-regular constraints (of (dis)equality constraints) applied is supposed to be bounded.

As shown in Example 5, the intersection of some horizontal CSs that are not necessarily WRRs, is still a horizontalin CS. So, is there a subclass of horizontal CSs, larger than WRRs, and closed under intersection?

References

1. A.C. Caron, F. Seynhaeve, S. Tison, and M. Tommasi. Deciding the Satisfiability of Quantifier Free Formulae on One-Step Rewriting. In *Proceedings of 10th Conference RTA, Trento (Italy)*, volume 1631 of *LNCS*. Springer-Verlag, 1999.
2. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA)*. http://l3ux02.univ-lille3.fr/tata.
3. L. Fribourg. SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting. In *proceedings IEEE Symposium on Logic Programming*, pages 172–185, Boston, 1985.
4. V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. In *Proceedings of FoSSaCs*, volume to appear of *LNCS*. Springer-Verlag, 2001.
5. M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176, 1997.
6. F. Jacquemard. *Automates d'Arbres et Réécriture de Termes*. Thèse de Doctorat d'Université, Université de Paris-sud, 1996. In French.
7. S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 429–440. Springer-Verlag, 1997. Full version in DMTCS (http://dmtcs.loria.fr/), volume 1, pages 69-98, 1997.
8. S. Limet and P. Réty. A New Result about the Decidability of the Existential One-step Rewriting Theory. In *Proceedings of 10th Conference on Rewriting Techniques and Applications, Trento (Italy)*, volume 1631 of *LNCS*. Springer-Verlag, 1999.
9. S. Limet, P. Réty, and H. Seidl. Weakly Regular Relations and Applications. Research Report RR-LIFO-00-17, LIFO, 2000. http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety/publications.html.
10. S. Limet and F. Saubion. On partial validation of logic programs. In M. Johnson, editor, *proc of the 6th Conf. on Algebraic Methodology and Software Technology, Sydney (Australia)*, volume 1349 of *LNCS*, pages 365–379. Springer Verlag, 1997.
11. S. Limet and F. Saubion. A general framework for R-unification. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *proc of PLILP-ALP'98*, volume 1490 of *LNCS*, pages 266–281. Springer Verlag, 1998.
12. J. Marcinkowski. Undecidability of the First-order Theory of One-step Right Ground Rewriting. In *Proceedings 8th Conference RTA, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.
13. J. Niehren, M. Pinkal, and P. Ruhrberg. On Equality up-to Constraints over Finite Trees ,Context Unification and One-step Rewriting. In W. Mc Cune, editor, *Proc. of CADE'97, Townsville (Australia)*, volume 1249 of *LNCS*, pages 34–48, 1997.
14. P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proceedings of the 6th international conference on Logic for Programming and Automated Reasoning (LPAR), Tbilisi (Republic of Georgia)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1999.
15. F. Seynhaeve, M. Tommasi, and R. Treinen. Grid Structures and Undecidable Constraint Theories. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 357–368. Springer-Verlag, 1997.
16. R. Treinen. The First-order Theory of One-step Rewriting is Undecidable. *Theoretical Computer Science*, 208:179–190, 1998.
17. S. Vorobyov. The First-order Theory of One-step Rewriting in Linear Noetherian Systems is Undecidable. In *Proceedings 8th Conference RTA, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.