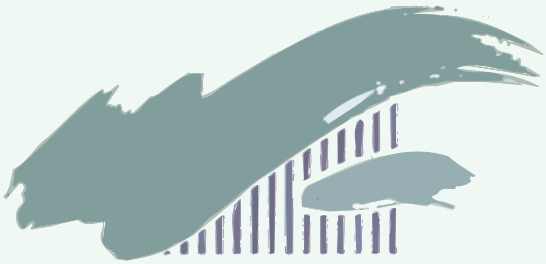


UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

Weakly Regular Relations and Applications

Sébastien Limet*, Pierre Réty*,
Helmut Seidl+

* LIFO - Université d'Orléans, France.
+ Dept. of Computer Science - University of Trier, Germany.

Rapport N^o **2000-17**

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

Submission Category : 1

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 by 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 RR's thanks to synchronization constraints between independent branches. TTSL's have first been introduced by means of *Tree-Tuple Synchronized Grammars (TTSG)* and have been applied to equational unification [7] and disunification [8], to logic program validation [10], and to one-step rewriting theory [9]. They have next been reformulated in a simpler way by means of *Constraint Systems (CS)*, and applied to rewriting again and to concurrency [4].

RR's have all usual nice properties[1], but expressiveness is poor. It is just the opposite for TTSL's. In particular, CS's are not closed by intersection[2], nor by complement.

In this paper we define an intermediate class, called *Weakly Regular Relations (WRR)*, between RR's and CS's, that has all RR's properties, except closure by complement. Thanks to its expressiveness greater than RR's, WRR's 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 instantiating by WRR's the general method [11] for deciding unifiability with the help of a tree-tuple language.
- on the existential one-step rewriting theory. Unlike [9], non-linear rewrite rules are allowed.

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

[2] Horizontal TTSG's are closed by intersection, provided a more complicated control is used [9], which makes great difficulties. In particular, we do not know if they are then still closed by 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 $\Sigma_\oplus = (\Sigma \cup \{\bot\}) \times (\Sigma \cup \{\bot\}) - \{\bot\bot\}$ where $\bot$ is a new constant. For $s,t \in T_\Sigma$ we recursively define $s \oplus t \in T_{\Sigma_\oplus}$ by:

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

For instance $f(a,g(b)) \oplus f(f(a,a),b)$ is $ff(af(\bot a,\bot a),gb(b\bot))$. 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 $f,g \in \Sigma$, $\{(t,t[f \leftarrow g]) \mid t \in T_\Sigma\}$ are RR's. On the other hand $\{(t,t_{sym}) \mid t \in T_\Sigma,\ t_{sym}$ is the symmetric of $t\}$ is not a RR.

2.2 Constraint Systems for Synchronized 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 confuse non-terminals with the sets they generate.

Example 2.1 *In the signature $\Sigma = \{f^{\setminus 2}, b^{\setminus 0}\}$ 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:*

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

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 2.2 *Now if we consider the slightly different CS :*

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

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

Example 2.3 *In the signature* $\Sigma = \{s^{\setminus 1}, b^{\setminus 0}\}$ *let* $L_{dble} = \{(s^n(b), s^{2n}(b))\}$*. It can be defined by the CS :*

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

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 2.1, $N = \{1,2\}$, $\mathcal{X} = \{Id_2\}$, $\rho(Id_2) = \{1,2\}$, 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 set $\mathcal{E}$ of inequations of the form

$$X \supseteq \Box(X_1, \ldots, X_k) \qquad (1)$$

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

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

denote the *disjoint union* of the index sets corresponding to the variables X_i (in example 2.1, $J = \{(1,1), (1,2), (2,1), (2,2)\}$ abbreviated into $\{1_1, 1_2, 2_1, 2_2\}$). Then $\Box$ 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 2.1. 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, $\Box$ is given by a tuple $\Box \in T_\Sigma(J)^{\rho(X)}$.

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 $\Box$;
- *horizontal* iff for any components $\Box_x$, $\Box_y$ and any positions p, q, $\Box_x|_p = (i, x')$, $\Box_y|_q = (i, y')$ implies p and q are positions at the same depth.
- *regular* iff each component of $\Box$ is a single constructor application of the form $\Box_x = a_x((1, x), \ldots, (n, x))$ for each $x \in \rho(X)$.

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

For example, the CS's 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 CS's is large enough to recognize synchronized languages. Therefore we consider only non-copying CS's in this paper.

Proposition 2.4 *[4] The class of non-copying CS's is closed by union, projection, cartesian product. Moreover membership and emptiness are decidable.*

Proposition 2.5 *RR's are exactly the languages defined by regular CS's.*

Proof 2.6 *If top symbols of all components of $\Box$ 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 $\rho(Y) = \{2\}$. Thus 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, $\rho(X) = \{1, 2, \ldots\}$.*

Note that regularity implies horizontality.

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

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

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 3.1 *Let L_1 and L_2 recognized by :*

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 makes a deadlock when computing the CS that would 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 3.2 *Emptiness of the intersection of synchronized languages is undecidable.*

Proof 3.3 *Let A and C be two alphabets and ϕ, ϕ' be morphism 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 $\phi(a_i) = c_{i,1} \dots c_{i,p_i}$ and $\phi'(a_i) = c'_{i,1} \dots c'_{i,p'_i}$ for each $a_i \in A$. Then L and L' are recognized by the following CS's[3] :*

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

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.

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

Considering the two previous examples, it seems reasonable 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 CS's.

Example 3.4 *Consider again languages Id_2 and L_{sym} as defined in Examples 2.1 and 2.2. So $Id_2 \cap L_{sym}$ is the set of pair of terms (t, t') such that $t = t'$ and t is the symmetric 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 of t. Let $X = \{(t, t_{sym}, t, t_{sym})\}$, $Id_2 \cap L_{sym}$ can be defined by*

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

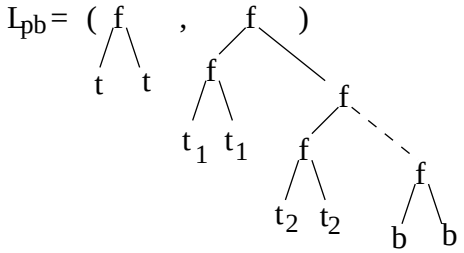
$$\begin{array}{rl} Id_2 \cap L_{sym} \supseteq & (b,b)|(f(1_1,1_2),f(1_3,1_4))X \\ X \supseteq & (f(1_1,2_1),f(2_2,1_2),f(1_3,2_3),f(2_4,1_4))(X,X)|(b,b,b,b) \end{array}$$

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 3.5 *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 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. See the appendix for a more formal justification.

That is why definition of WRR's 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 $r(X)$ horicontal but non-regular constraints.

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

Example 4.1 *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 $S = \{(t_1,t_2) \mid t_1 \to_R t_2\} = \{C[l\sigma], C[r\sigma]\}$. S is a WRR :*

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

$$S \supseteq (f(1_1,2_1),g(2_2,1_2))(Id_2,Id_2)$$

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

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

Fact 4.2
- *If the constraint system $\mathcal{C}$ is weakly regular, then it is weakly regular for a rank function with maximal rank $2^{|\mathcal{C}|}$.*
- *It can be decided in polynomial (even 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 (missing proofs are in the appendix):

Theorem 4.3 *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 4.1 defines a WRR indeed, since every recursive rule is regular and Id_2 is a RR. On the other hand, L_{sym} (Example 2.2) is not a WRR.

Our main new result is:

Theorem 4.4 *Weakly regular relations are closed under union, projection, cartesian product and intersection.*

The first three closure properties of WRR's stated in theorem 4.4 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

$$X \supseteq \Box(X_1)$$

The proof of theorem 4.4 is based onto the following two auxiliary theorems:

Theorem 4.5 *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.

Theorem 4.6 *Assume that the tree-tuple language L of class I is defined by a single-permuting constraint system $\mathcal{C}$ and $\sim \subset I \times I$ 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 theorems 4.5 and 4.6, the proof of theorem 4.4 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 theorem 4.5, 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 theorem 4.6, we can construct from $\mathcal{C}'$ a constraint system $\mathcal{C}''$ which defines $L^{\sim}$. Finally, we may (due to closure by projections) 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 4.4, we obtain:

Corollary 4.7 *Weakly regular relations are closed under joins.*

Proof 4.8 *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 by 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 $\Sigma = \{f^{\setminus 2}, a^{\setminus 0}\}$. Thus the set of unbalanced full binary trees is the set of unbalanced terms over Σ, which is generated by the following constraint system (V is the axiom):

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

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 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 TTSG's [7]. Next the method has been generalized [11] : any tree-tuple language can be used, provided :

1. it is closed by 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 by intersection[4] and projection, the goal to be unified may be non-linear.

Thus, the work using TTSG's looks as an instance of this generalization, and another instance using primal grammars [5] has been achieved [11] and provides a decidability result under other restrictions. In this section we present another instance, by using WRR's, whose restrictions are those of TTSG's, except that the goal[5] and the non-recursive rewrite rules may be non-linear, plus[6] some others (see Theorem 5.12 for details).

Let us first recall the principle of the general method, using a simple example :

Example 5.1

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

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 :

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

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

From narrowing properties [3], there exists a data-unifier iff there exist instances t, t' such that $r_1 = r'_1$, 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', r_1, r'_1, and force equalities by intersection. Thus, there exists a data-unifier iff $\Pi_{1,3}(N) \cap \Pi_{1,4}(N') \neq \emptyset$.

[4] This implies the closure by join.

[5] WRR's are closed by intersection and projection.

[6] Additional restrictions are necessary, otherwise unifiability is undecidable, as shown in [7].

For each narrowing step occurring within a narrowing derivation $f(x) \leadsto^*_{[\sigma]} r$ 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 then $f(t) = t$ because $f(f(t)) = t$. Therefore $t = 0$.

Using WRR's

$\boldsymbol{N_c}$ **:** unfortunately, the sets $N_c = \{(c(t_1,\ldots,t_n), t_1,\ldots,t_n)\}$ cannot be expressed by WRR's (nor by RR's), 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 are 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 :

$$\begin{aligned} N_c &\supseteq (c(1_1,\ldots,n_1), \natural 1_2,\ldots,\natural n_2)(Id_2,\ldots,Id_2) \\ \natural L &\supseteq (\natural 1_1,\ldots,\natural 1_n)(L) \end{aligned}$$

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 5.2 *Consider the previous example again. Now :*

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

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

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 5.3 *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[7] *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 $l \to r$) is weak-horizontal *if $t \doteq t'$ (resp. $l \to r$) is a weak-horizontal term, considering $\doteq$ (resp. $\to$) is a binary symbol.*

Lemma 5.4 *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 $\|x\|$ times $\natural$ exactly.*

Proof 5.5 *When using n-ary symbols, the intermediate language N^u corresponding to position u ($N^\epsilon = N$) is*

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

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

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

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 5.6 *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 5.7 *Function calls in the term t are* shallow *if for all function position p in t, $t|_p = f(x_1, \dots, x_n)$ where $x_1, \dots, x_n$ are variables.*

Definition 5.8 *Given a rewrite system R, we define an ordering on defined function by $f > g$ if $f(t_1, \dots, t_n) \to r \in R$ and g occurs in r. $f \equiv g$ means $f > g \wedge g > f$. The rewrite rule $f(t_1, \dots, t_n) \to r \in R$ is* recursive *if there is a function g in r s.t. $f \equiv g$.*

[7] This definition of weak-horizontality is a little different from that of Section 6 because only constructors matter here.

Algorithm : we assume that function calls in rhs's are shallow, and recursive rewrite rules are linear. Let us write rewrite rules as follow $f(t_1, \ldots, t_n) \to C[u_1, \ldots, u_m]$ where C is a constructor context containing no variables and each u_i is either a variable or a function call of the form $f_i(x^i_1, \ldots, x^i_{k_i})$. 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 $u_i = f_i(x^i_1, \ldots, x^i_{k_i})$ 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_i's are defined independently of F. So we can make some variables equal by computing the intersection[8] 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.

Example 5.9 *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 5.10 *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 5.11 *A constructor-based rewrite system is* weak-regular *if function calls 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 WRR's, we get:

Theorem 5.12 *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.*

[8] if one of the F_i's is not a WRR, the algorithm fails.

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 $\rightarrow$, where $x \rightarrow y$ means x rewrites into y by one step.

It has been shown undecidable in [17]. Sharper undecidability results have been obtained for some subclasses of rewrite systems, about the $\exists^*\forall^*$-fragment [16, 12] and the $\exists^*\forall^*\exists^*$-fragment [18].

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][9], for the positive existential fragment [13], for the whole existential fragment in the case of quasi-shallow[10] rewrite systems [1] and also in the case of linear, non-overlapping, non-ϵ-left-right-overlapping[11] rewrite systems [9].

Thanks to WRR's, we get a new result about the existential fragment.

Definition 6.1 *A rewrite system R is* ϵ-left-right-clashing *if for all rewrite rule $l \rightarrow r$, $l(\epsilon) \neq r(\epsilon)$. R is* weakly-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 6.2 *The existential one-step rewriting theory is decidable in the case of ϵ-left-right-clashing weakly-horizontal rewrite systems.*

Since quasi-shallowness is a particular case of weak-horizontality, our result extends that of [1], except that we assume in addition ϵ-left-right-clashing. Compared to [9], rewrite rules may now be non-linear and overlapping, but they must be weak-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 $x \rightarrow x$ or $x \rightarrow y$. Because of ϵ-left-right-clashing, $x \rightarrow x$ has no solutions and is replaced by the predicate without solutions $\bot$, and $x \rightarrow y$ is replaced by the equivalent proposition $(x \stackrel{?}{\rightarrow}_{[ru_1]} y \vee \ldots \vee x \stackrel{?}{\rightarrow}_{[ru_n]} y) \wedge x \neq y$, where $\stackrel{?}{\rightarrow}_{[ru_i]}$ 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 $x \stackrel{?}{\rightarrow}_{[ru_i]} y$ 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 $C = \ x \stackrel{?}{\rightarrow}_{[ru_1]} y \wedge \neg(y \stackrel{?}{\rightarrow}_{[ru_2]} z)$. The solutions of $x \stackrel{?}{\rightarrow}_{[ru_1]} y$ (denoted $SOL(x \stackrel{?}{\rightarrow}_{[ru_1]} y)$) and $SOL(\neg(y \stackrel{?}{\rightarrow}_{[ru_2]} z))$ are WRR's of pairs, then we can compute $SOL(C) = \ SOL(x \stackrel{?}{\rightarrow}_{[ru_1]} y)) \times T_\Sigma \ \cap \ T_\Sigma \times SOL(\neg(y \stackrel{?}{\rightarrow}_{[ru_2]} z))$, which is still a WRR (of triples).

[9] Even the theory of several-step rewriting is decidable.

[10] All variables in the rewrite rules occur at depth one.

[11] I.e. no left-hand-side overlaps on top with the corresponding right-hand-side.

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

$SOL(x = y)$ and $SOL(x \neq y)$ are trivially WRR's since they are RR's.

Lemma 6.3 *$SOL(x \xrightarrow{?}_{[ru_i]} y)$ and $SOL(\neg(x \xrightarrow{?}_{[ru_i]} y))$ are WRR's.*

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 WRR's, still closed by intersection (for applications), could extend the result of [14] by weakening the restrictions.

Compared to to automata with (dis)equality constraints [2], WRR's can define more constraints than only (dis)equality, but they cannot define the balanced terms. However, WRR's 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 3.4, the intersection of some horizontal CS's that are not necessarily WRR's, is still a horizontal CS. So, is there a subclass of horizontal CS's, larger than WRR's, and closed by 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. Research Report 2000-16, LIFO, 2000. http://www.univ-orleans.fr/SCIENCES/LIFO. Submitted.

[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. Solving Disequations modulo some Class of Rewrite Systems. In *Proceedings of 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japon)*, volume 1379 of *LNCS*, pages 121–135. Springer-Verlag, 1998.

[9] 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.

[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] H. Seidl and A. Neumann. On Guarded Nested Fixpoints. In *Annual Conference of the European Association of Logic in Computer Science (CSL)*, volume 1683 of *LNCS*, pages 484–498. Springer-Verlag, 1999.

[16] 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.

[17] R. Treinen. The First-order Theory of One-step Rewriting is Undecidable. In H. Ganzinger, editor, *Proceedings 7th Conference RTA, New Brunswick (USA)*, volume 1103 of *LNCS*. Springer-Verlag, 1996.

[18] 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.

Appendix

A Balanced terms are not generated by a CS

Lemma A.1 *The language of balanced terms over the signature $\{f^{/2}, a^{/0}\}$ is not recognized by constaint system with finite number of variables.*

Proof A.2 *Let us suppose that sc is a constraint system with a finite number of variables. Let us call X_{init} the axiom of sc.*

Since the number of variables of sc is finite and each variable of sc defines a tuple with a finite number of components, there exists only a finite number of contextes C, such that C does not contain the constant a and $X_{init} \supseteq C[u_1, \ldots, u_k](X)$ where X is a single variable of sc. So to get all the balanced terms we get one of these two situations:

- *$X_{init} \supseteq C[u_1, \ldots, u_k](X)$ where C contains at least one constant a. In this case the depth of the generated balanced term is fixed which determines uniquely the term so X recognizes a sigleton language. Since the language of balanced terms is infinite, we should have an infinite number of variable X, which contradict the hypothesis.*
- *$X_{init} \supseteq C[u_1, \ldots, u_k](X_1, \ldots, X_n)$ with $n > 1$. So each variable X_i recognizes a tuple tp_i such that $C[u_1, \ldots, u_k](tp_1, \ldots, tp_n)$ is a balanced term. But if at least one of these variable (say X_j) recognizes another tuple tp'_j, $C[u_1, \ldots, u_k](tp_1, \ldots tp'_j, \ldots, tp_n)$ which is not a balanced term, is also recognized by sc .So all the X_i's recognize singleton languages, so sc needs an infinity of variables*[12]*.*

So if sc contains only a finite number of variables, it does not recognized the set of balanced terms.

B Proof of Theorem 4.3

Assume $\mathcal{C}$ is weakly regular with rank function r. Consider a recursive constraint $X \supseteq \Box(X_1, \ldots, X_k)$. If $r(X) = 0$, then the constraint must be regular. Since $r(X) \geq r(X_1) + \ldots + r(X_k)$, all variables X_i have ranks 0 as well. Hence, all constraint systems $\mathcal{C}_{X_i}$ are regular.

Now assume $r(X) > 0$. Since the constraint is recursive, the set R of all X_j depending on X is non-empty. By definition of the rank function, we find that

$$r(X) = r(X_j) \qquad \text{for all } X_j \in R$$

As $r(X)$ exceeds 0 and also cannot be smaller than the sum of the non-negative values $R(X_i)$, we conclude that R consists of just one element, say X_i, where $r(X_j) = 0$ for all

[12] We do not make any hypothesis about C which may or not contain the constant a.

$i \neq j$. Thus, $\mathcal{C}_{X_j}$ is regular for all $i \neq j$ – which completes the proof of the "only-if" direction of the theorem.

For the reverse implication, we are going to construct a rank function according to the definition of weakly regular constraint systems where, in particular, the rank of X is 0 iff $\mathcal{C}_X$ is regular. We proceed by induction on the number d of strong components in the variable dependence graph.

If $d = 1$, there is just one strong component S, and all constraints $X \supseteq \square(X_1, \ldots, X_k)$ with $k > 0$ are recursive. By assumption, all these constraints are regular implying that the whole constraint system is regular. Hence, we may assign rank 0 to all $X \in S$.

Now assume $d > 1$. Let S denote a strong component such that for every constraint with left-hand side X, we always have $X \in S$ already provided some variable of the right-hand side is in S. In particular, we obtain a constraint system $\mathcal{C}'$ with $d - 1$ strong components if we remove from $\mathcal{C}$ all variables from S and all constraints with left-hand sides in S. By induction hypothesis, $\mathcal{C}'$ is weakly regular, and we can construct an appropriate rank function for the variables of $\mathcal{C}'$. It remains to determine a value M as the common rank of all variables in S.

Let us assign a value $r[c]$ to each constraint c with left-hand side in S such that any $M \geq r[c]$ will satisfy the conditions of weak regularity for the constraint c. The goal then is to define M as the least upper bound on all these values $r[c]$.

Let c be the constraint $X \supseteq \square(X_1, \ldots, X_k)$. If c is non-recursive, then none of the variables X_j is in S. In case c is regular, we then set $r[c]$ to the sum of ranks of all X_j. Otherwise, we set $r[c]$ to the sum of ranks of the X_j plus 1. Clearly, this definition meets our requirements.

If c is recursive, we set $r[c] = 0$. Let us prove that this choice is indeed sufficient. As c is recursive, we know by assumption, that c is regular. Moreover, at least one X_i is contained in S. If $\mathcal{C}_{X_i}$ is regular, then also $\mathcal{C}_{X'}$ must be regular for all $X' \in S$. Hence, M has to be chosen as 0 – complying with our definition of $r[c]$. If on the other hand, $\mathcal{C}_{X_i}$ is non-regular, then all the constraint systems $\mathcal{C}_{X_j}$ with $i \neq j$ must be regular, and any choice for the ranks of X and X_i will satisfy the assumptions of weak regularity for c – as long as they are equal.

This finishes the proof.

C Proof of Theorem 4.5

Let $\mathcal{C}$ denote a weakly regular constraint system. The key idea for removing non-regular constraints which are not single-permuting consists in creating new variables which simulate the joint cartesian product of all successors. Here, we define the cartesian product of constraints

$$X^{(j)} \supseteq \square^{(j)}(X_1^{(j)}, \ldots, X_m^{(j)}), \qquad j = 1, \ldots, n$$

as the constraint

$$Z \supseteq \square(Z_1, \ldots, Z_m)$$

where

- m equals the maximum of the $m^{(j)}$;
- $Z = X^{(1)} \dots X^{(n)}$ is a variable corresponding to the sequence of left-hand sides $X^{(j)}$ of class $\rho Z = \{(j, x) \mid x \in \rho X^{(j)}\}$; moreover, Z receives the rank $r(X^{(1)}) + \dots + r(X^{(n)})$;
- accordingly, Z_i is a variable corresponding to the sequence of all i-th successors $X_i^{(j)}$ where $i \leq m^{(j)}$ of class $\rho Z_i = \{(j, x) \mid i \leq m^{(j)}, x \in \rho X_i^{(j)}\}$;
- $\Box_{(j,x)} = \Box^{(j)} \tau_j$ where τ_j maps the projections (i, y) to $(i, (j, y))$.

Now we proceed by induction on the maximal rank r of a variable having a non-regular constraint (or 0 – if no such variable exists).

If $r = 0$, nothing has to be done. Therefore, assume $r > 0$. Then we replace all constraints

$$X \supseteq \Box(X_1, \dots, X_m)$$

where X is of rank r and $m > 1$, by the constraint

$$X \supseteq \Box'(Z)$$

where Z equals the sequence $X_1 \dots X_m$ (classified as $\{(i, x) \mid x \in \rho X_i\}$) and $\Box'$ is obtained from $\Box$ by replacing the projections (i, x) with the projections $(1, (i, x))$.

Moreover, we add the cartesian product of all m-tuples of constraints where the sums of ranks of left-hand sides are at most $r - 1$. The resulting constraint system $\mathcal{C}'$ has the following properties:

1. $\mathcal{C}$ and $\mathcal{C}'$ are equivalent (up to the auxiliary new variables in $\mathcal{C}'$);
2. The maximal rank of a variable having a non-regular constraint with more than one successor is at most $r - 1$;
3. If n, d, a are the size, the maximal size of a class and the maximal number of successors in right-hand sides of rules of $\mathcal{C}$, then $n^{a+1}, d \cdot a$ and a are the corresponding numbers for $\mathcal{C}'$.

This finishes the proof.

D Proof of Theorem 4.6

Let $\mathcal{C} = (\mathcal{X}, N, \mathcal{E})$ be a constraint system where each non-regular constraint is single permuting. Furthermore, assume we are given a variable X classified as I and an equivalence relation $\sim$ on I. Our goal is to construct a new constraint system $\mathcal{C}'$ having a variable $X^{\sim}$ such that the tuples of $X^{\sim}$ are precisely given by the tuples of X which satisfy the equality constraints specified by $\sim$. In order to do so, we construct $\mathcal{C}' = (\mathcal{X}', N, \mathcal{E}')$ where the new set of constraint variables $\mathcal{X}'$ consists of all pairs $X^{\sim}, X \in \mathcal{X}$, $\sim$ equivalence relation on ρX. Moreover, each constraint

$$X \supseteq \Box(X_1, \dots, X_m) \qquad (*)$$

gives rise to all the constraints

$$X^{\sim} \supseteq \Box(X_1^{\sim_1}, \ldots, X_m^{\sim_m})$$

where $\sim$ and $\sim_1, \ldots, \sim_m$ are equivalence relations on the respective classes such that the following holds:

- If the constraint $(*)$ is regular, then whenever $i \sim j$ and $\Box_i = a((1,i), \ldots, (k,i))$ and $\Box_j = a'((1,j), \ldots, (k',j))$ then $a = a'$ (and hence also $k = k'$) and $i \sim_s j$ for all $s = 1, \ldots, k$.
- If the constraint $(*)$ is non-regular (and hence $m = 1$), then whenever $i \sim j$ and $\Box_i = a((1,i_1), \ldots, (1,i_k))$ and $\Box_j = a'((1,j_1), \ldots, (1,j_{k'}))$ then $a = a'$ (and hence also $k = k'$) and $i_s \sim_1 j_s$ for all $s = 1, \ldots, k$.

All these constraints together form the set $\mathcal{E}'$. Let σ and σ' denote the least solutions of the constraint systems $\mathcal{C}$ and $\mathcal{C}'$. By fixpoint induction, we verify for every $X \in \mathcal{X}$ and equivalence relation $\sim$ on ρX:

$$\sigma'(X^{\sim}) = \{t \in \sigma(X) \mid \forall (i,j) \in \sim\colon t_i = t_j\}$$

this finishes the proof.

E Proof of Lemma 6.3

E.1 $SOL(x \xrightarrow{?}_{[ru_i]} y)$ is a WRR

Let $ru_i = l \to r$, $Var(l) = \{x_1, \ldots, x_n\}$, and for each x_i let $|x_i|$ be the number of occurrences of x_i in the dummy term $c(l,r)$, and $p_i^1, \ldots, p_i^{|x_i|}$ be their positions. The terms l', r' are defined by $c(l',r') = c(l,r)[p_i^j \leftarrow i_j, \forall i, \forall j]$.

For a positive integer k, let $Id_k = \{(t, \ldots, t) \mid t \in T_\Sigma \wedge length(t, \ldots, t) = k\}$. Id stands for Id_1 and do not confuse Id_k with the cartesian product $Id^k = Id \times \ldots \times Id$.

For $f \in \Sigma$, let $ar(f)$ be the arity of f, $A_f = \{1, \ldots, ar(f)\}$ and $\vec{Y}_f[i \leftarrow X]$ denotes $(Y, \ldots, Y, X, Y, \ldots, Y)$ where X is the i^{th} component and the tuple length is $ar(f)$. f will be omitted when no confusion.

Consider the following contraint system:

$$\begin{array}{rcll} \bigcup_{f \in \Sigma} \cup_{i \in A_f} X & \supseteq & (f(1_1, \ldots, ar(f)_1),\ f(1_2, \ldots, ar(f)_2))I\vec{d}_2[i \leftarrow X] & (1) \\ X & \supseteq & (l', r')(Id_{|x_1|}, \ldots, Id_{|x_n|}) & (2) \end{array}$$

(1) generates any context, in which a rewrite step will possibly be applied (2). Therefore this constraint system generates $SOL(x \xrightarrow{?}_{[l \to r]})$ indeed. On the other hand, (1) is recursive, but it is regular and Id_2 is a RR. (2) is not regular, but it is not recursive. Moreover (1) is horizontal, and the weak-horizontality of $l \to r$ implies the horizontality of (2). Consequently (1) $\cup$ (2) is a WRR.

E.2 $SOL(\neg(x \stackrel{?}{\rightarrow}_{[ru_i]} y))$ is a WRR

$SOL(x \stackrel{?}{\rightarrow}_{[l\rightarrow r]} y)$ is the least fixpoint of the constraint system given in the previous proof. To negate a constraint, one should negate either the operator or any of the arguments. Therefore $SOL(\neg(x \stackrel{?}{\rightarrow}_{[l\rightarrow r]}))$ is the greatest fixpoint of the following constraint system:

$$\begin{array}{lll} \bar{X} \subseteq & \bigcup_{f\neq g\in\Sigma}(f(Id,\ldots,Id),\, g(Id,\ldots,Id)) & (1)\\ & \bigcup_{f\in\Sigma}(f(1_1,\ldots,ar(f)_1),\, f(1_2,\ldots,ar(f)_2)) & \\ & \qquad \left(\bigcap_{i\in A_f}(\cup_{j\neq i}\vec{Id}^2[j\leftarrow I\bar{d}_2]\cup \vec{Id}^2[i\leftarrow \bar{X}])\right) & (2)\\ \bar{X}\subseteq & Y & (4)\\ & \bigcup_{i=1,\ldots,n}(l',r')(Id^{|x_1|},\ldots,\bar{Id}_{|x_i|},\ldots,Id^{|x_n|}) & (5) \end{array}$$

where Y generates $\bar{l}\times\bar{r}$ where $\bar{l}$ (resp. $\bar{r}$) denotes the set of terms that clash with l (resp. r), i.e. $\bar{l}=\{t\in T_\Sigma \mid \exists p\in \overline{Pos}(l), t(p)\neq l(p)\}$. Thus Y is the cartesian product of two regular tree languages, then Y is a RR.

Now X generates $SOL(x \stackrel{?}{\rightarrow}_{[l\rightarrow r]} y)$, which includes the case of zero-step rewriting, i.e. the pairs of identical terms. Then necessarily $X\supseteq Id_2$, then $\bar{X}\subseteq I\bar{d}_2\subseteq Id^2$. Thanks to this property, (2) is equivalent to:

$$\begin{array}{ll} \bigcup_{f\in\Sigma}\cup_{i\in A_f}(f(1_1,\ldots,ar(f)_1),\, f(1_2,\ldots,ar(f)_2))\, \vec{Id}^2[i\leftarrow\bar{X}] & (2')\\ \bigcup_{f\in\Sigma}\cup_{i\neq j\in A_f}(f(1_1,\ldots,ar(f)_1),\, f(1_2,\ldots,ar(f)_2))\, \vec{Id}^2[i,j\leftarrow I\bar{d}_2] & (3') \end{array}$$

Since we are only interested in the greatest fix-point, we can write:

$$\bar{X} = \big((1)\cup(2')\cup(3')\big)\cap((4)\cup(5))$$

Now, since $l(\epsilon)\neq r(\epsilon)$, we have $(5)\cap(2')=\emptyset$, $(5)\cap(3')=\emptyset$, $(5)\subseteq(1)$, $(2')\subseteq(4)$, $(3')\subseteq(4)$. Therefore

$$\bar{X} = ((1)\cap(4))\cup(2')\cup(3')\cup(5)$$

Note that (1) is the cartesian product of two regular tree languages, then it is a RR, then $(1)\cap(4)$ is a RR. Let Z be a non-terminal that generates $(1)\cap(4)$. On the other hand, the only recursive constraint comes from (2'), which is guarded. Then from Theorem 5 in [15], there is only one fix-point. Therefore $SOL(\neg(x \stackrel{?}{\rightarrow}_{[l\rightarrow r]} y))$ is the least fix-point of the constraint system:

$$\{\bar{X}\supseteq Z,\, \bar{X}\supseteq(2'),\, \bar{X}\supseteq(3'),\, \bar{X}\supseteq(5)\}$$

Z is a RR. Since $I\bar{d}_2$ is a RR, (2') and (3') are RR. (5) is not regular, but it is not recursive. Consequently the whole constraint system is a WRR.