Grid Structures and Undecidable Constraint Theories*

Franck Seynhaeve[1], Marc Tommasi**[1], Ralf Treinen[2]

[1] LIFL, Bât M3, Université Lille 1, F59655 Villeneuve d'Ascq cedex, France
email: {seynhaev,tommasi}@lifl.fr, Web: http://www.lifl.fr/~tommasi
[2] LRI & CNRS, Bât. 490, Université de Paris-Sud, F91405 Orsay cedex, France
email: treinen@lri.fr, Web: http://www.lri.fr/~treinen

Abstract. We express conditions for a term to be a finite grid-like structure. Together with definitions of term properties by excluding "forbidden patterns" we obtain three new undecidability results in three areas: the $\exists^*\forall^*$-fragment of the theory of one-step rewriting for linear and shallow rewrite systems, the emptiness for automata with equality tests between first cousins (i.e. only tests at depth 2 below the current node are available), and the $\exists^*\forall^*$-fragment of the theory of set constraints.

1 Introduction

Domino games and Turing machines are well-known tools to prove undecidability results. The grid structure provides convenient means for encoding computation sequences of Turing machines. In its infinitary version (i.e. $\mathbb{Z}\times\mathbb{Z}$), it has been used for instance to obtain undecidability results for monadic second order theories [21,19,6,14,15]. A classical encoding of the computation of a Turing Machine can be done only with a local matching on a grid, where, roughly speaking, row i contains a description of the tape at time i, and column j contains the values of cell j of the tape during the computation. Only local tests are necessary to verify that successive rows in the grid correspond to successive tapes in a successful computation of the machine.

In this paper we prove undecidability results for computational mechanisms over *finite terms*. Intuitively, a term is a grid-like term if from each node, going one step up and then one step to the right yields the same subterm than going one step to the right and then one step up. In other words, the directed acyclic graph associated with a grid-like term is a grid.

Basically, the common approach for the results we prove here is the following. We have to express two properties: that a term is a grid-like term, and that the grid encodes a computation of a Turing machine. Since the latter can be done using local tests only, regular tools such as rewrite systems or tree automata can be used to exclude certain "forbidden" patterns.

Using these techniques we prove that the following theories are undecidable:

* Partially supported by The Esprit working group CCL II (22457) and the HCM project CONSOLE (CHRXCT940495)
** Partially supported by "GDR AMI" Groupement De Recherche 1116 du CNRS

To appear in: Proceedings of the 1997 Colloquium on Trees in Algebra and Programming (CAAP'97), Lille, France, April 1997, Springer LNCS

– the $\exists^*\forall^*$-fragment of the theory of one step rewriting for the class of shallow and linear rewrite system, and
– the emptiness property for tree automata with equality tests between first cousins, and
– the $\exists^*\forall^*$-fragment of the theory of set constraints.

One-step rewriting The theory of one-step rewriting for a given rewrite system R and signature Σ is the first-order theory of the following structure: its universe consists of all Σ-ground terms, and its only predicate is the relation "x rewrites to y in one step by R". The structure contains no function symbols and no equality. In [23] it has been shown that there is no algorithm which decides the $\exists^*\forall^*$-fragment of the theory of one-step rewriting for any rewrite system R. This result has been refined to the $\exists^*\forall^*$-fragment for the class of *linear* rewriting systems in [22], to the $\exists^*\forall^*$-fragment for the class of *right ground* rewriting systems in [16] and to the $\exists^*\forall^*\exists^*$-fragment for the class of *linear noetherian* rewriting systems in [24]. Recently, decidability of the positive existential fragment has been shown in [12].
In this paper we restrict the class of rewriting systems for which the theory of one-step rewriting is undecidable to the class of *linear and shallow* term rewriting systems. This undecidability result is surprising in the light of the *decidability result* for the quotient algebra by a finite set of shallow equations [5].

Tree automata with equality tests Tree automata with equality tests have been introduced by Dauchet and Mongy to tackle non-linearity problems in various fields such as rewriting, program approximation, and partial evaluation [17]. On the one hand, the class of languages recognized by tree automata with equality tests is closed under non linear morphisms and classical boolean operations. On the other hand, when unrestricted equalities are allowed the emptiness property for these acceptors is undecidable. This negative result stems from the fact that equalities can be propagated in a term as far as desired using transitivity of the equality and repeated application of non-linear rules. In the original paper, the authors encode the Post correspondence problem using overlapping equalities between subterms at different depth.
When only equalities between direct subterms (brothers) are allowed it is not possible to overlap equalities, and Bogaert and Tison have shown that in this case the emptiness problem is decidable [2].
Closely related, Caron et al [3] have defined encompassment automata, that is automata with equality tests which can handle a bounded number of equalities along each path of a tree and between brothers in an unrestricted way. They generalized the result of [2] because the emptiness problem is decidable for encompassment automata.
Consequently, one could hope to keep decidability while testing equalities (and disequalities) at the same depth. However, we prove in this paper that the emptiness problem for Tree Automata with equality tests between First Cousins – Trafic-automata – is undecidable.

Set constraints Set constraints are relationships between sets of terms of a Herbrand Universe. Because of their expressive power and their naturalness, they have been used in program analysis [9,11]. The main idea is to associate with a program variable an approximation of the set of its possible values. Set constraints have also enriched (constraint) logic programming languages, in order to compute with sets [13].

Relations between automata and set constraints have been first pointed out by Heintze and Jaffar in [10]: the case of set constraints between sets of words can be treated using a translation into monadic second order logic of k successors, i.e. Rabin tree automata [18]. In [7] this approach is reused and a new class of tree automata which can handle the case of set (of terms) constraints is defined. As an advantage, tree automata provide for decision algorithms and closure properties [8].

In a more general way, we can examine the satisfiability problem for formulas of a theory based on set constraints denoted by $\mathcal{T}_{\mathrm{SC}}$. The language of this theory is defined in the following way: atomic formulas are *elementary set constraints* of the form $t \subseteq t'$; formulas are obtained from atomic formulas by closure under boolean operators (and, or, not) and quantifiers. More precisely, the syntactic definition of atomic formulas relies on a set of variables $\mathcal{X}$ and a finite set Γ of functions symbols. Then, an elementary set constraint is of the form $t \subseteq t'$ where $t, t' \in T_\Gamma(\mathcal{X})$. An interpretation $\mathcal{I}$ of a set constraint maps each variable of $\mathcal{X}$ onto a subset of T_Γ.

The complete theory $\mathcal{T}_{\mathrm{SC}}$ is undecidable because of the undecidability of the monadic theory of finitely generated free algebras [20] and the existential fragment is decidable [1,4,7]. This paper states that the satisfiability problem for formulas of the $\exists^*\forall^*$-fragment is undecidable.

The paper is organized as follows. In Section 2.1 we explain how local grid-patterns can be used in order to describe compution sequences of a Turing machine. In Section 2.2 we introduce finite grid-like terms. The definition has to take care of the borders of the grid. This implies a special treatment at the leaves of the terms. The encoding of the halting problem is then presented in Section 3 for rewrite systems and Section 4 for tree automata. Finally, the undecidability result for set constraints is given in Section 5.

2 Preliminaries

Let $T_\Gamma(\mathcal{X})$ denote the set of terms over a ranked alphabet Γ and a set $\mathcal{X}$ of variables, and let $t \in T_\Gamma(\mathcal{X})$. We denote by $Var(t)$ the set of variables which occur in t and by $t|_p$ the subterm of t rooted at position p . We have $head(t) = \alpha$ iff $t(0) = \alpha$, that is α is the root symbol of t.

2.1 Turing Machines and Computations

For the rest of the paper we fix an instance of a restricted class of Turing machines: let $T = (Q, I, q_s, q_f)$ be a Turing machine with tape alphabet $\{a, b\}$ ($\square$ is the blank symbol), state set $Q = \{q_1, \ldots, q_k\}$, initial state q_s, accepting state q_f and instruction set I. We can assume w.l.o.g.

- that T never accesses a tape position to the left of the starting position,

- that it never writes a $\Box$ on the tape (hence, $\Box$ can never occur to the left of the head).

The signature Σ as well as several other entities to be constructed during the proof depend on the Turing machine T. For the sake of brevity we do not mention the index T which strictly speaking is in order here.

We specify the configuration of the Turing machine by a string called *instantaneous description*. As usual, the configuration is noted by concatenating the part of the tape left to the head, the state, and the part of the tape starting at the head position (such that the tape symbol seen by the head is written to the immediate right of the state). For technical reasons, we will in addition delimit the string by the \$ start mark and the # stop mark; the stop mark is always preceding by the blank symbol and furthermore the symbols on the left half of the tape will always be indexed with l, while the symbols on the second half are indexed with r.

Definition 1 (Instantaneous Description). We define the following sets of constants:

$\langle \mathit{leftchar}\rangle := a_l \mid b_l$ $\quad\quad \langle \mathit{state}\rangle := q_1 \mid \ldots \mid q_k$

$\langle \mathit{rightchar}\rangle := a_r \mid b_r \mid \Box_r$ $\quad \langle \mathit{constant}\rangle := \$ \mid \# \mid \langle \mathit{leftchar}\rangle \mid \langle \mathit{rightchar}\rangle \mid \langle \mathit{state}\rangle$

An *instantaneous description (ID)* is a string licensed by the following regular expression: $\langle \mathit{id}\rangle := \$\langle \mathit{leftchar}\rangle^*\langle \mathit{state}\rangle\langle \mathit{rightchar}\rangle^+\Box_r\#$

Definition 2 (P_{id}). The set P_{id} is the following set of patterns where _ matches any character:

$\_\$ \mid \$\langle \mathit{rightchar}\rangle \mid \$\# \mid \langle \mathit{leftchar}\rangle\langle \mathit{rightchar}\rangle \mid \langle \mathit{leftchar}\rangle\# \mid \langle \mathit{state}\rangle\langle \mathit{leftchar}\rangle \mid$
$\langle \mathit{state}\rangle\langle \mathit{state}\rangle \mid \langle \mathit{state}\rangle\# \mid \langle \mathit{rightchar}\rangle\langle \mathit{leftchar}\rangle \mid \langle \mathit{rightchar}\rangle\langle \mathit{state}\rangle \mid \#\_ \mid a_r\# \mid b_r\#$

Lemma 3. *A string $w \in \langle \mathit{constant}\rangle^*$ is an instantaneous description if w starts with \$, ends with #, and none of the patterns of P_{id} matches w.*

A sequence of IDs can be stored in the upper right quarter of an infinite plane partitioned into cells (recall that a Turing machine's tape is left bounded) where each line corresponds to an ID. We detail now conditions for such a plane (or grid) to be a computation by means of 2-dimensional patterns.

Definition 4 (P_T). The set P_T is the following set of patterns:

$\begin{array}{c} x \\ \hline \$ \end{array}$ where $x \neq \$$.

$\begin{array}{c|c} x & y \\ \hline \# & \end{array}$ where $x \neq \#$ and $(x \neq \Box_r$ or $y \neq \#)$.

$\begin{array}{c|c} x & \\ \hline c_l & u \end{array}$ where $u \notin \langle \mathit{state}\rangle$ and $x \neq c_l$, $c_l \in \langle \mathit{leftchar}\rangle$

$\begin{array}{c|c} & x \\ \hline u & c_r \end{array}$ where $u \notin \langle \mathit{state}\rangle$ and $x \neq c_r$, $c_r \in \langle \mathit{rightchar}\rangle$

$\begin{array}{c|c|c} x & y & z \\ \hline u & q & c_r \end{array}$ where $(q, c) \mapsto (p, d, 0)$ and $(x \neq u$ or $y \neq p$ or $z \neq d_r)$.

$\begin{array}{c|c|c} x & y & z \\ \hline u & q & c_r \end{array}$ where $(q, c) \mapsto (p, d, R)$ and $(x \neq u$ or $y \neq d_l$ or $z \neq p)$.

$\begin{array}{c|c|c} x & y & z \\ \hline u & q & c_r \end{array}$ where $(q, c) \mapsto (p, d, L)$ and $(x \neq p$ or $y \neq u$ or $z \neq d_r)$.

Lemma 5. *A grid g represents a computation if the first line of g is the initial configuration $\$ q_s \Box_r \#$ and none of the patterns of P_{id} or P_T matches g.*

2.2 Terms Representing Grids

Definition 6 (Signature Γ). The signature Γ consists of the ternary symbol f and the constants $\perp_0, a_l, b_l, a_r, b_r, \Box_r, \$, \#, q_1, \ldots, q_k$.

The symbols of Γ are used to represent computation sequences of the machine T. Note that the tape symbols come in two variants: left and right handed.

Definition 7 (Γ-grid). A ground term t over some super-signature of Γ is called a Γ-grid if

1. $t \in T_\Gamma$;
2. for every subterm $f(x, y, z)$ we have
   (a) $x = \perp_0$ or $head(x) = f$,
   (b) $y = \perp_0$ or $head(y) = f$ and
   (c) $z \in \{a_l, b_l, a_r, b_r, \Box_r, \$, \#, q_1, \ldots, q_k\}$;
3. for each subterm $f(f(x_1, y_1, z_1), f(x_2, y_2, z_2), z_3)$
   (a) the equation $y_1 = x_2$ holds and
   (b) $head(y_1) = f$;
4. for each subterm $f(x_1, f(f(x_2, y_2, z_2), y_3, z_3), z_1)$, $head(x_1) = f$.

Hence, the directed acyclic graph of a Γ-grid t is a grid in the sense of the last section. In a term t, the last argument of an f-term is the content of a cell, the first argument is the upper neighbour, and the second argument is the neighbour to the right. The "end" of the grid (on a row or on a column) corresponds to a leaf $\perp_0$ of t. The last three conditions need some explanations:
Condition 3a states that by going one step up and then one step to the right one gets the same description than by going first one step to the right and then one step up. Condition 3b states that when there is a description of the upper neighbour and of the right neighbour of some cell, then there is also a description of the upper right neighbour. Consequently, every $i+1$-th row is as least as long as the i-th row. Finally, condition 4 states that if a cell has an upper-right neighbour then it has an upper neighbour, too. Consequently, all lines start at the same position (see Figure 1).

3 Rewriting

3.1 Preliminaries

A rewrite system is called shallow, resp. linear, if all its rules are shallow, resp. linear. A rewrite rule $l \to r$ is called shallow, resp. linear, if both l and r are shallow, resp. linear. A term t is *shallow* if all its variables occur at depth at most one. A term t is *linear* if it does not contain any multiple variable occurrences. We employ the following abbreviations:

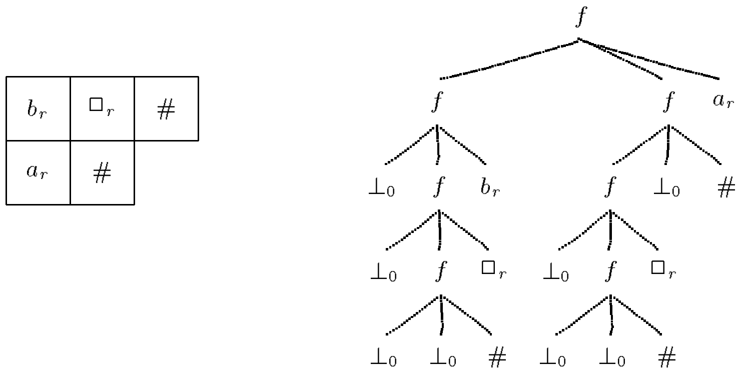


Fig. 1. A grid of two lines and its corresponding Γ-grid t.

$$x \to^1 y := x \to y \qquad\qquad x \to^{n+1} y := \exists z\, (x \to z \wedge z \to^n y)$$
$$x \Rightarrow y := x \to y \wedge y \not\to x \qquad\qquad x \Rightarrow^{n+1} y := \exists z\, (x \Rightarrow z \wedge z \Rightarrow^n y)$$
$$x \Rightarrow^1 y := x \Rightarrow y$$

The set of leaf positions of a term t is denoted by $\mathcal{LP}os(t)$, and its set of non-leaf positions by $\mathcal{IP}os(t)$. The set of all positions of t is $\mathcal{P}os(t) = \mathcal{LP}os(t) \cup \mathcal{IP}os(t)$. The implication sign of predicate logic is written $\supset$ in order to distinguish it from the rewrite relation symbol $\to$.

Definition 8. Let Σ be a signature and R be a Σ-rewrite system. The structure $\mathcal{A}_{\Sigma,R}$ is defined as follows: The language of $\mathcal{A}_{\Sigma,R}$ contains no constants or function symbols, and its only predicate symbol is the binary predicate symbol $\to$. The universe of $\mathcal{A}_{\Sigma,R}$ is the set $T(\Sigma)$, and $t \to s$ holds in $\mathcal{A}_{\Sigma,R}$ iff t rewrites to s in one rewriting step of R.

3.2 The Undecidability Proof

Theorem 9. *There is no algorithm which decides for any signature Σ and any linear and shallow Σ-rewrite system R the $\exists^*\forall^*$-fragment of the theory of $\mathcal{A}_{\Sigma,R}$.*

We are going to reduce the halting problem for the restricted class of Turing machines defined in Section 2.1 to the validity of a certain formula in some structure $\mathcal{A}_{\Sigma,R}$.

Definition 10 (Signature Σ, rewrite system R). The signature Σ is the extension of Γ by the constants u, r, u', r'. The Σ-rewrite system R consists of the following rules (note that R is shallow and linear):

$$f(x,y,z) \to u(x) \mid r(y) \mid u'(x) \mid r'(y) \qquad u'(x) \to r(x) \qquad r'(x) \to u(x)$$

Lemma 11. *For every finite set P of linear terms in $T_\Sigma(\mathcal{X})$ there exist a signature extension $\Sigma_P \supseteq \Sigma$, a shallow and linear Σ_P-rewrite system $R_P \supseteq R$, a quantifier-free formula $ad(x)$ and for every $p \in P$ an $\exists^*$-formula $match[p](x)$ such that:*

- *for every* $t \in T_{\Sigma_P}$*:* $\mathcal{A}_{\Sigma_P,R_P} \models ad(t)$ *iff* $t \in T_\Gamma$*;*
- *for every* $p \in P$ *and* $t \in T_\Sigma$*:* $\mathcal{A}_{\Sigma_P,R_P} \models match[p](t)$ *iff* p *matches* t*.*

Proof. Let $P = \{t_1, \ldots, t_n\}$. We define

$$\Sigma_P := \Sigma \cup \{c_{t,o} \mid t \in P, o \in \mathcal{IP}os(t)\} \cup \{e_{i,j} \mid 1 \le i \le n, 1 \le j \le n+i-1\}$$

For any $t \in P$ and $o \in \mathcal{P}os(t)$, let $d_{t,o} := t|_o$ if $o \in \mathcal{LP}os(t)$, and $d_{t,o} := c_{t,o}$ if $o \in \mathcal{IP}os(t)$.

$$\begin{aligned} R_P := {} & R \cup \{h(\bar{z}) \to h(\bar{z}) \mid h \notin \Gamma\} \\ & \cup\ \{f(d_{t,o1}, \ldots, d_{t,op}) \to c_{t,o} \mid t \in P, o \in \mathcal{IP}os(t), head(t|_o) = f, arity(f) = p\} \\ & \cup\ \{d_{t_i,\epsilon} \to e_{i,1}, e_{i,j} \to e_{i,j+1}, e_{i,n+i-1} \to d_{t_i,\epsilon} \mid 1 \le i \le n, 1 \le j < n+i-2\} \end{aligned}$$

Finally, we define, where $t_i \in P$ and k is the cardinality of $\mathcal{IP}os(t_i)$:

$$\begin{aligned} ad(x) &:= x \not\to x \\ match[t_i](x) &:= \exists y\, (x \to^k y \wedge y \Rightarrow^{n+i-1} y) \end{aligned}$$

Definition 12 ($grid(x)$).

$$\begin{aligned} grid(x) := {} & ad(x) \wedge \bigwedge_{h \notin \{\perp_0, f\}} \Big(\neg match[f(h(\_), \_, \_)](x) \ \wedge\ \neg match[f(\_, h(\_), \_)](x)\Big) \\ & \wedge\ \neg match[f(\_, \_, f(\_))](x) \ \wedge\ \neg match[f(\_, \_, \perp_0)](x) \\ & \wedge\ \forall y, z, z', v \Big(x \to^2 y \ \wedge\ match[f(r(\_), u'(\_), \_)](y) \ \wedge\ y \to z \\ & \qquad \wedge\ match[u(r(\_))](z) \ \wedge\ y \to z' \ \wedge\ match[r'(u'(\_))](z') \\ & \qquad \wedge\ z' \to v \ \wedge\ match[r'(r(\_))](v)\Big) \ \supset\ v \to z \\ & \wedge\ \bigwedge_{h \neq f} \neg match[f(f(\_, h(\_), \_), f(\_, \_, \_), \_)](x) \\ & \wedge\ \bigwedge_{h \neq f} \neg match[f(h(\_), f(f(\_, \_, \_), \_, \_), \_)](x) \end{aligned}$$

Note that $grid(x)$ is a $\forall^*$ formula since each occurrence of a *match*-formula is negated.

Lemma 13. *Let* $\Sigma_P \supseteq \Sigma$ *and* $R_P \supseteq R$ *be constructed according to Lemma 11, where* P *contains all the patterns mentioned in Definition 12. Then a term* $t \in T(\Sigma_P)$ *is a* Γ*-grid iff* $\mathcal{A}_{\Sigma_P,R_P} \models grid(t)$*.*

Definition 14 ($init$).

$$\begin{aligned} init(x) := {} & match[f(\_, f(\_, f(\_, f(\_, \perp_0, \#), \Box_r), q_s), \$)](x) \\ & \wedge \neg match[f(f(\_, f(\_, f(\_, f(\_, \perp_0, \#), \Box_r), q_s), \$), \_, \_)](x) \end{aligned}$$

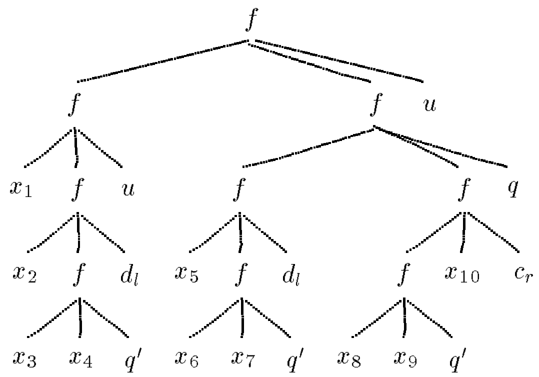
The first part of $init(g)$ where g is a Γ-grid states that some row of the grid g starts with $\$q_s\square_r\#$, that is with the instantaneous description of the initial configuration. The second part states that there is no preceding line to the initial configuration (there might be some line that is not an instantaneous description). Note that $init(x)$ is a $\exists^*\forall^*$-formula.

Each two-dimensional pattern on the grid can be expressed as a term of $T_\Gamma(X)$.

Definition 15. We associate with each pattern of P_{id} and P_T a term of $T_\Gamma(\mathcal{X})$. This yields sets P'_{id} and P'_T of terms of $T_\Gamma(\mathcal{X})$.

We illustrate the last definition with an small example of a pattern p of P_T and its associated term $T_\Gamma(\mathcal{X})$:

u	d_l	q'
u	q	c_r



Definition 16. We define the following formulae:

$$\begin{aligned} trans(x) &:= \bigwedge_{p\in P'_T} \neg match[p](x) \wedge \bigwedge_{p\in P'_{id}} \neg match[p](x) \\ final(x) &:= match[q_f](x) \\ halts &:= \exists x \left(grid(x) \wedge init(x) \wedge trans(x) \wedge final(x) \right) \end{aligned}$$

Proof of Theorem 9. Let $\mathcal{A}_{\Sigma,R} \models grid(t) \wedge init(t) \wedge trans(t)$. Then t represents a computation sequence.
Let M be the set of all patterns used in the above constructions, and let Σ_M and R_M be according to Lemma 11. Then the Turing machine T halts on the empty input iff $\mathcal{A}_{\Sigma_M,R_M} \models halts$. This completes the proof of Theorem 9.

4 Automata

Definition 17. A *Trafic*-automaton is a 4-tuple $\mathcal{A} = (\Gamma, E, F, R)$ where Γ is a finite ranked alphabet; E is a finite set of unary letters called *states* disjoint from Γ; $F \subseteq E$ is a set of final states; and R is a finite set of *transition rules*, all being of the following two forms, where $q_1, \dots, q_n, q \in E$ and $x_i, x_{i_j}^k \in \mathcal{X}$:

$$f(q_1(x_1), \dots, q_n(x_n)) \to q$$
$$f(q_1(f_1(x_1^1, \dots, x_{p_1}^1)), \dots, q_n(f_n(x_1^n, \dots, x_{p_n}^n))) \to q.$$

Note that rules may be non-linear, that is the same variable may occur twice or more often in a left hand side.
The tree language recognized by a *Trafic*-automaton $\mathcal{A}$ is the set of terms of T_Γ which are reduced by the rewrite system R into a final state:

$$L(\mathcal{A}) = \{t \in T_\Gamma \mid t \overset{*}{\mapsto} q(t) \text{ where } q \in F\}.$$

The class of languages recognized by *Trafic*-automata is closed under union and intersection.
Let us recall that the class of languages recognized by tree automata are closed under union, intersection and complementation. Let us remark that languages recognized by tree automata are also recognized by *Trafic*-automata.

Theorem 18. *The emptiness problem in the class of Trafic-automata is undecidable.*

We are going to reduce the halting problem for the restricted class $\mathcal{MT}$ of Turing machines defined in Section 2.1 to the emptiness problem in the class of *Trafic*-automata. In other words, given a Turing machine T, we build a *Trafic*-automaton $\mathcal{A}_{halts}$ such that $\mathcal{L}(\mathcal{A}_{halts})$ encodes all successful computations of T.
In the proof, we first state in Lemma 19 that Γ-grids are recognizable by Trafic-automata. Then, Lemma 20 proves that codes of a successful computation are also recognizable by Trafic-automata.

Lemma 19. *There exists a Trafic-automaton $\mathcal{A}_{grid}$ such that $\mathcal{L}(\mathcal{A}_{grid}) = \{t \mid t \text{ is a } \Gamma\text{-grid}\}$.*

Proof. Let us consider Γ_0 the set of constants of Γ and the *Trafic*-automaton $\mathcal{A}_g = (\Gamma, \{q\}, \{q\}, R)$ whose rules are:

$$\begin{array}{rrr}
\forall a, b, c \in \Gamma_0 \quad a \to q & f(q(a), q(b), q(c)) \to q & f(q(a), q(b), C) \to q \\
f(q(a), B, q(c)) \to q & f(q(a), B, C) \to q & f(A, q(b), q(c)) \to q \\
f(A, q(b), C) \to q & f(A, B, q(c)) \to q & f(A, B, C) \to q \\
\textit{with } A = q(f(x_1, y_1, z_1)) & B = q(f(x_2, y_2, z_2)) & C = q(f(x_3, y_3, z_3))
\end{array}$$

We prove that $\mathcal{L}(\mathcal{A}_g) = \{t \in T_\Gamma \mid t \textit{ satisfies Condition 3a of Definition 7}\}$ using induction on the structure of terms for the $\supseteq$ part and using induction on the number of derivations for the $\subseteq$ part.

Conditions 2, 3b and 4 of Definition 7 correspond to local tests on subterms and hence $\{t \in T_\Gamma \mid t \textit{ satisfies Conditions 2, 3b and 4 of Definition 7}\}$ is a regular language, then recognizable by a (classical) tree automaton. Then we can construct a tree automaton $\mathcal{A}_{gl}$ such that:

$$\mathcal{L}(\mathcal{A}_{gl}) = \{t \in T_\Gamma \mid t \textit{ satisfies Conditions 2, 3b and 4 of Definition 7}\}.$$

Moreover $\mathcal{L}(\mathcal{A}_g) \cap \mathcal{L}(\mathcal{A}_{gl}) = \{t \mid t \textit{ is a } \Gamma\textit{-grid}\}$. Finally, since every tree automaton is also a trafic-automaton and the class of *Trafic*-automata is closed under intersection, there exists a *Trafic*-automaton $\mathcal{A}_{grid}$ such that $\mathcal{L}(\mathcal{A}_{grid}) = \mathcal{L}(\mathcal{A}_g) \cap \mathcal{L}(\mathcal{A}_{gl})$.

Lemma 20. *Let T be a Turing machine in $\mathcal{MT}$. There exists a Trafic-automaton $\mathcal{A}_{halts}$ such that $\mathcal{L}(\mathcal{A}_{halts}) = \{$successful computations of the Turing machine $T\}$.*

Proof. Let T be a Turing machine. For any term p of P'_{id} and P'_T, p is linear then $\{t \in T_\Gamma \mid \textit{some instance of } p \textit{ is a subterm of } t\}$ is a regular langage.

Moreover $\{t \in T_\Gamma \mid t|_0 = f(u_1, f(u_2, f(u_3, f(u_4, \bot_0, \#), \Box_r), q_s), \$)\}$ and $\{t \in T_\Gamma \mid f(\bot_0, u, q_f) \textit{ is a subterm of } t\}$ are also regular langages. Then we define in the same way than in Section 3.2 (classical) tree automata $\mathcal{A}_{init}$, $\mathcal{A}_{trans}$, $\mathcal{A}_{final}$ and $\mathcal{A}_{halts}$ such that:

$$\begin{aligned}\mathcal{L}(\mathcal{A}_{halts}) &= \mathcal{L}(\mathcal{A}_{grid}) \cap \mathcal{L}(\mathcal{A}_{init}) \cap \mathcal{L}(\mathcal{A}_{trans}) \cap \mathcal{L}(\mathcal{A}_{final}) \\ &= \{ \textit{ successful computations of } T\}.\end{aligned}$$

The Turing machine T halts on the empty input iff $\mathcal{L}(\mathcal{A}_{halts}) \neq \emptyset$. This completes the proof of Theorem 18.

5 Theory of Set Constraints

Theorem 21. *There is no algorithm which decides for any ranked alphabet Γ the $\exists^*\forall^*$-fragment of $\mathcal{T}_{SC}$.*

The proof first relies on Lemma 24 which states that there exists a formula $grid(X)$ in the $\exists^*\forall^*$-fragment of $\mathcal{T}_{\mathrm{SC}}$ such that $grid(X)$ is satisfiable if and only if X denotes a singleton set containing a Γ-grid. Then, following the construction of Section 3, in Lemma 25 we build a formula *halts* which is satisfiable if and only if the machine T halts on the empty input.

Definition 22. For the sake of brevity, we define the emptyset $\emptyset$ and the intersection $\cap$ as follows:

$$\begin{aligned}\emptyset &\equiv \forall X\ \emptyset \subseteq X \\ X \cap Y = \emptyset &\equiv \forall Z\ (Z \subseteq X \wedge Z \subseteq Y) \Rightarrow Z = \emptyset\end{aligned}$$

Definition 23. Let $C = \Gamma_0 \setminus \{\bot_0\}$ and $grid$(X) be a formula defined as follows:

$$grid(X) \equiv sing(X) \wedge \big(\exists S\ subterms(X, S) \wedge shape(S) \wedge equalities(S) \wedge edge(S)\big)$$

where

$$\begin{aligned}
sing(X) &\equiv X \neq \emptyset \wedge (\forall Y\ Y \subseteq X \Rightarrow X \subseteq Y \vee Y \subseteq \emptyset) \\
subterms(X, S) &\equiv S \subseteq f(S, S, C) \cup \bot_0\ \wedge\ X \subseteq S \\
&\quad \wedge \forall S'\ (S' \subseteq f(S', S', C) \cup \bot_0 \wedge X \subseteq S') \Rightarrow S \subseteq S' \\
shape(S) &\equiv \forall Y_1, Y_2, Y_3, \bar{Z}\ \ f(f(Y_1, \bot_0, Y_2), f(\bar{Z}), Y_3) \cap S = \emptyset \\
equality(S) &\equiv \forall Y_1, Y_2, Y_3, Y_4, Y_5, Y_6 \\
&\quad (\textstyle\bigwedge_i Y_i \neq \emptyset\ \wedge\ f(f(Y_1, Y_2, Y_3), f(Y_4, Y_5, Y_6), Y_7) \subseteq S) \Rightarrow Y_2 = Y_4 \\
edge(S) &\equiv \forall Y_1, Y_2, Y_3, Y_4, Y_5, Y_6\ \ f(\bot_0, f(f(Y_1, Y_2, Y_3), Y_4, Y_5), Y_6) \cap S = \emptyset
\end{aligned}$$

Lemma 24. $\forall X\ (grid(X) \quad \Leftrightarrow \quad \exists t\ \Gamma\text{-}grid\ st\ X = \{t\})$

Proof. The formula $sing(X)$ is satisfied iff X is a singleton. Let $t \in X$. The formula $subterms(X, S)$ defines the set S of subterms of t except the elements of C. The first line of the formula defines a subterm-closed set containing t. For example, let t in S ($t \neq \bot_0$):

$$t \in S \Rightarrow t \in f(S, S, C) \Rightarrow \exists t_1, t_2, t_3\ st\ t = f(t_1, t_2, t_3) \Rightarrow t_1, t_2 \in S,\ t_3 \in C.$$

The second one defines S as the smallest set satisfying the first line.
Finally, the formulae $shape(S)$, $equality(S)$ and $edge(S)$ translate Conditions 2, 3 and 4 of Definition 7.

Lemma 25. *There exists a formula* halts *of the* $\exists^*\forall^*$*-fragment of* $\mathcal{T}_{SC}$ *such that* halts *is satisfiable iff the Turing machine* T *halts on the empty input.*

Proof. Let T be a Turing machine and X such that $grid(X)$. Then, according to Lemma 24, there exists a Γ-grid t such that $X = \{t\}$. $grid(X)$ is a formula of the $\exists^*\forall^*$-fragment of $\mathcal{T}_{\mathrm{SC}}$, and it defines the set S of all subterms of the term t. We define now in a same way than in Section 3.2 formulas $init(S)$, $trans(S)$, $final(S)$ and $halts$. To this aim, we just note that for any term p of P'_{id} and P'_T, we can say that p matches or does not match a term in S:

$$match[p](S) \ \equiv\ \exists \bar{u}\ p \subseteq S \wedge p \neq \emptyset$$

where $\bar{u}$ denotes the set of variables of the term p. The preceding formula is satisfied if there exists set of terms $\mathcal{I}(x_1) \ldots, \mathcal{I}(x_l)$ ($\mathcal{I}$ interpretation and $x_1, \ldots, x_l$ variables of p) such that $\mathcal{I}(p) \subseteq \mathcal{I}(S)$ and $\mathcal{I}(p) \neq \emptyset$.

The Turing machine T halts on the empty input iff $halts$ is satisfiable. This completes the proof of Theorem 21.

Acknowledgement

We thank Sophie Tison for her useful comments and corrections, and Nachum Dershowitz and Joachim Niehren for pleasant discussions.

References

1. A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30–44, Oct. 1995.
2. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Lectures Notes in Computer Science*, volume 577, pages 161–171, Paris, 1992. Symposium on Theoretical Aspects of Computer Science.
3. A. Caron, H. Comon, J. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Lectures Notes in Computer Science*, volume 820, pages 436–449, Jerusalem, july 1994. 21st international colloquium on Automata, Languages and Programming.
4. W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In IEEE, editor, *Proc. 35th Symp. Foundations of Computer Science*, pages 642–653, 1994.

5. H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, cycle-syntacticness and shallow theories. *Information and Computation*, 111(1):154–191, May 1994.
6. B. Courcelle. The monadic second-order logic of graphs I. recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
7. R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the* 34^{th} *Symp. on Foundations of Computer Science*, pages 372–380, 1993. Full version in the LIFL Tech. Rep. IT-247.
8. R. Gilleron, S. Tison, and M. Tommasi. Some new decidability results on positive and negative set constraints. In *Proceedings, First International Conference on Constraints in Computational Logics*, volume 845 of *LLNCS*, pages 336–351, 1994.
9. N. Heintze. *Set Based Program Analysis*. PhD thesis, Carnegie Mellon University, 1992.
10. N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, Philadelphia, Pennsylvania, 4–7 June 1990. IEEE Computer Society Press.
11. N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proceedings of the* 17^{th} *ACM Symp. on Principles of Programming Languages*, pages 197–209, 1990. Full version in the IBM tech. rep. RC 16089 (#71415).
12. P. R. Joachim Niehren, Manfred Pinkal. On equality up-to constraints over finite trees, context unification, and one-step rewriting, Dec. 1996.
13. D. Kozen. Logical aspects of set constraints. In *Proceedings of Conf. Computer Science Logic (CSL'93)*, volume 832 of LNCS, pages 175–188, 1993.
14. H. Läuchli and C. Savioz. Monadic second order definable relation on the binary tree. *Journal of Symbolic Logic*, 52:219–226, 1987.
15. H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.
16. J. Marcinkowski. Undecidability of the first-order theory of one-step right ground rewriting. To appear in RTA'97, 1997.
17. J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.
18. M. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
19. D. Seese. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53:169–195, 1991.
20. M. Taitslin. Some further examples of undecidable theories. *Algebra and Logic*, 7:127–129, 1968. Original paper (russian) in Algebra i Logica 6(3):105-111, 1967.
21. W. Thomas. On logics, tilings, and automata. In *Annual International Colloquium on Automata, Languages and Programming*, 1991.
22. R. Treinen. The first-order theory of one-step rewriting by a linear term rewriting system is undecidable (extended abstract), June 1996.
23. R. Treinen. The first-order theory of one-step rewriting is undecidable. In H. Ganzinger, editor, *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 276–286, New Brunswick, NJ, USA, July 1996. Springer-Verlag.
24. S. Vorobyov. The first-order theory of one step rewriting in linear noetherian systems is undecidable. To appear in RTA'97, 1997.