Grid Structures and Undecidable Constraint Theories*

Franck Seynhaeve Sophie Tison
Marc Tommasi
LIFL, Bât M3, Université Lille 1, F59655 Villeneuve d'Ascq cedex, France
email: lastname@lifl.fr, Web: http://www.lifl.fr/~tommasi

Ralf Treinen
LRI & CNRS, Bât. 490, Université Paris-Sud, F91405 Orsay cedex, France
email: treinen@lri.fr, Web: http://www.lri.fr/~treinen

September 28, 1999

Abstract

We prove three new undecidability results for computational mechanisms over finite trees: There is a linear, ultra-shallow, noetherian and strongly confluent rewrite system R such that the $\exists^*\forall^*$-fragment of the first-order theory of one-step-rewriting by R is undecidable; the emptiness problem for tree automata with equality tests between cousins is undecidable; and the $\exists^*\forall^*$-fragment of the first-order theory of set constraints with the union operator is undecidable.

The common feature of these three computational mechanisms is that they allow us to describe the set of first-order terms that represent grids. We extend our representation of grids by terms to a representation of linear two-dimensional patterns by linear terms, which allows us to transfer classical techniques on the grid to terms and thus to obtain our undecidability results.

1 Introduction

The grid structure provides convenient means for encoding computation sequences of Turing machines. A classical encoding of the computation of a Turing machine can be achieved just 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 its infinitary version (i.e. $\mathbb{Z} \times \mathbb{Z}$), the grid has been used, for instance, to obtain undecidability results for monadic second order theories [29, 24, 11, 18, 19]. The reader is referred to Börger *et al.* book [4] for further reading on tiling, dominoes, grids and (un)decidability.

In this paper we prove undecidability results for computational mechanisms over *finite terms*. Turing computations are encoded in so-called grid-terms. Roughly speaking, a grid-term is a term representation of a grid structure. Basically, it must

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

fulfill a translation of the grid property: from any cell, going first up and then right gives the same cell than going first right and then up.
Hence, the common approach for the results we prove here is the following. In any computational mechanism we investigate, we have to express two properties: that a term is a grid-term, and that the grid encodes a computation of a Turing machine. Since the latter can be done using local tests only, usual techniques in rewriting theory or tree automata theory can be used to exclude certain "forbidden patterns".
Using these techniques we prove that the following problems are undecidable:

- There is a linear, ultra-shallow, Noetherian and strongly confluent rewrite system for which the $\exists^*\forall^*$-fragment of the theory of one-step rewriting is undecidable.
- The emptiness problem in the class of tree automata with equality tests between cousins is undecidable.
- The $\exists^*\forall^*$-fragment of the first-order theory of set constraints is undecidable.

One-step rewriting

The theory of one-step rewriting for given rewrite systems $R_1, \ldots, R_p$ and signature Σ is the first-order theory of the following structure: its universe consists of all Σ-ground terms, and its only predicates are the relation "x rewrites to y in one step by R_i". In [30] it has been shown that the $\exists^*\forall^*$-fragment of this theory is undecidable. This result has been refined to the $\exists^*\forall^*$-fragment for the class of *linear* rewrite systems in [31], to the $\exists^*\forall^*$-fragment for the class of *right ground* and for the class of linear Noetherian rewrite systems in [20] and to the $\exists^*\forall^*\exists^*$-fragment for the class of *linear Noetherian* rewrite systems in [32, 33].
In this paper we restrict the class of rewrite systems for which the theory of one-step rewriting is undecidable to the class of *linear, ultra-shallow and convergent* term rewriting system, a rule being ultra-shallow if variables occurs at depth one. This undecidability result is surprising in the light of the decidability result for the first order theory of the quotient algebra $T_\Gamma/_{=_E}$ when Γ is a finite signature and E a finite set of shallow equations [10]. Hence, for any shallow rewriting system, the theory of the symmetric transitive closure of the one-step rewriting relation is decidable.
Decidability of the positive existential fragment has been shown in [22], for any rewrite system. In [6], it is proved that the existential fragment of one-step rewriting is decidable for ultra-shallow rewrite systems. Hence crossing the border between decidability and undecidability just consists in adding universal quantifiers, leading to a formula with one quantifier alternation.
A part of the construction we use in this paper for checking some properties of terms is very similar to the one used by Vorobyov in [32].

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 [21]. 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 [21], 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 [3].
Closely related, Caron *et al* [5] 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 [3] in showing that the emptiness problem is decidable for encompassment automata.
Consequently, one could hope to keep decidability while testing equalities (and inequalities) 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.
As a consequence, we also prove that it is undecidable whether a language recognizable by Trafic automaton is regular, that is recognizable by some finite tree automaton.

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 [14, 16]. 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 [17].
Relations between automata and set constraints have been first pointed out by Heintze and Jaffar in [15]: 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 [23]. In [13] 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 [12].
In a more general way, we can examine the satisfiability problem for formulas of the first-order theory based on set constraints denoted by $\mathcal{T}_{\mathrm{SC}}$. Terms are built using set variables and function symbols. An *atomic set constraint* is of the form $t \subseteq t'$ where t, t' are terms. An interpretation $\mathcal{I}$ of a set constraint maps each variable of $\mathcal{X}$ into a subset of ground terms. Set operators as $\cup$ (union), $\cap$ (intersection), $\sim$ (complement) can enrich the language.
The theory $\mathcal{T}_{\mathrm{SC}}$ is undecidable because of the undecidability of the monadic second order theory of trees [27], while the existential fragment of this theory with the set operators $\cup, \cap, \tilde{}$ is decidable [13, 8, 1]. In [26], the undecidability result is refined to the $\exists^*\forall^*\exists^*$-fragment of the theory. In this paper we show that the satisfiability problem for formulas of the $\exists^*\forall^*$-fragment is undecidable as soon as the union set operator is allowed. Recently, Talbot and Charatonik have independently improved this result by removing the union operator [28, 7].
The undecidability result for set constraints is interesting regarding the class of automata defined in [13]. In [13], a class of languages recognizable w.r.t. a new kind of automata — GTSA (for Generalized Tree Set Automata) is introduced and is proved to be closed under projection (*i.e.* existential quantification) but not closed under complement (*i.e.* negation). The class of GTSA-recognizable languages contains the existential fragment of $\mathcal{T}_{\mathrm{SC}}$. Hence, the result stated in this paper shows that it is not possible to design a class of automata with "good" properties, *i.e.* for which the emptiness problem is decidable, closed by complement

and projection, which recognize the existential fragment of $\mathcal{T}_{\mathrm{SC}}$ (hence, the set of solutions of systems of set constraints).

For convenience, the grid encoding is given with a signature that contains one ternary letter. One ternary letter avoids introducing many binary ones and simplifies proofs, essentially for the rewrite system presented in Section 4. All the results in this paper also hold with a similar encoding but restricted to only binary letters and constants (see Remark 6 for a simple explanation). Of course, the situation changes when the signature contains only constants and unary function symbols: the emptiness problem of automata with equality tests, the theory of one-step rewriting and the first-order theory of set reduce to WS2S and are hence decidable.

The paper is organized as follows. In Section 3 we explain how local grid-patterns can be used in order to describe computation sequences of a Turing machine and we introduce grid-terms. The encoding of the halting problem is then presented in Section 4 for rewrite systems and Section 5 for tree automata. Finally, the undecidability result for set constraints is given in Section 6.

2 Preliminaries

An alphabet Γ is *ranked* if $\Gamma = \bigcup_n \Gamma_n$ where each Γ_m is a finite set and $\Gamma_n \neq \emptyset$ only for a finite number of n's. Elements of Γ_n are said to be of *arity* n. Elements of arity 0, 1, ..., n are respectively called *constants*, *unary*, ..., *n-ary* symbols. In the following, f/n will denote an element of Γ_n. A ranked alphabet is also called a signature.

We assume that Γ contains at least one constant. Let $\mathcal{X}$ be a set of variables and let $\mathbb{N}^*$ denote the set of finite–length strings over $\mathbb{N}$.
A *term* (considered here as a labelled tree) over $\Gamma \cup \mathcal{X}$ is a partial function $t : \mathbb{N}^* \to \Gamma \cup \mathcal{X}$ with domain $\mathcal{P}os(t)$ satisfying the following properties:

- $\mathcal{P}os(t)$ is nonempty and prefix-closed.
- If $t(\alpha) \in \Gamma_n$, then $\{i | \alpha i \in \mathcal{P}os(t)\} = \{0, 1, \dots, n-1\}$.
- If $t(\alpha) \in \mathcal{X}$, then $\{i | \alpha i \in \mathcal{P}os(t)\} = \emptyset$.

The set of all terms (or *trees*) is denoted by $T_\Gamma(\mathcal{X})$. If $\mathcal{X} = \emptyset$ then $T_\Gamma(\mathcal{X})$ is also written as T_Γ and terms of T_Γ are said *ground*. Each element of $\mathcal{P}os(t)$ is called a *position*. Two positions p_1, p_2 of $\mathcal{P}os(t)$ are said *comparable* if there exists p such that $p_1 = pp_2$ or $p_2 = pp_1$, otherwise they are said *incomparable* or *parallel*. If $p_1 = p_2 p$ then p_2 is a prefix of p_1 and we write $p_2 \trianglelefteq p_1$.

Let $t \in T_\Gamma(\mathcal{X})$. We denote by $\mathrm{Var}(t)$ the set of variables which occur in t, by $t|_p$ the subterm of t rooted at position p and by $t(p)$ the label of t at position p. We write $head(t)$ for $t(\epsilon)$, that is the root symbol of t.
A term $t \in T_\Gamma(\mathcal{X})$ is said *linear* if every variable occurs at most once in t. Let $m \in \mathbb{N}$ and $\mathcal{X}_m$ denoting the set of variables $\{x_1, \dots, x_m\}$. If t is a linear term of $T_\Gamma(\mathcal{X})_m$, then $t[t_1, \dots, t_m]$ denotes the term obtained by replacing each variable x_i by a term t_i. If $m = 1$, t is called a *context*.

A *substitution* σ is a mapping from $\mathcal{X}$ into $T_\Gamma(\mathcal{X})$ which differs from the identity for a finite number of variables only. We extend a substitution σ to $T_\Gamma(\mathcal{X})$ in the following way:

$$\forall f \in \Gamma_n, \forall t_1, \dots, t_n \in T_\Gamma(\mathcal{X}), \sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n)).$$

A term $t \in T_\Gamma(\mathcal{X})$ is called an *instance* of a term $s \in T_\Gamma(\mathcal{X})$ if there exists a substitution σ such that $\sigma(s) = t$. The instance t is said ground if t is ground.
Finally a term $u \in T_\Gamma(\mathcal{X})$ *matches* t if there exists a substitution σ such that $\sigma(u)$ is a subterm of t.

3 Turing machine Computations and Grids

In this section we develop the tool that we are going to use for proving our main results. All undecidability results will be obtained by reduction of halting problems of Turing machines (either for a Turing-complete class of machines on the empty tape, or for a universal machine on an arbitrary tape). Turing machine computations will be represented on finite grids where the cells hold state symbols or tape symbols of a Turing machine.

We start by establishing a representation of finite grids by a special class of terms called *grid-terms*. Then, we show how two-dimensional patterns on a finite grid translate to term-patterns on grid-terms. Finally we summarize the two-dimensional patterns that characterize the grids holding a Turing machine computation. Together with the correspondence between grids and grid-terms (and there respective notions of patterns) this will yield a representation of Turing machine computations by grid-terms.

3.1 Grids and grid-terms

We think of a grid as set of cells (u, v) where u is the row, v the column and where the origin $(0, 0)$ is the lower left corner.

Grids are used to encode computation sequences of Turing machines: row i of a grid contains a description of the tape at time i, and column j contains the values of cell j of the tape during the computation.

Note that an assumption like "a pattern matches the grid" can be seen as a "local" property (it suffices to have one position in the grid where the pattern matches) while an assumption "a pattern does not match a grid" expresses a global property of the grid, since at *all* positions of the grid the pattern must not match. The property that all rows of the grid must be obtained from their predecessor by a computation of a Turing machine is of global nature, hence we can only express it by saying that some patterns do *not match*. There is, however, one difficulty: With assertions that some patterns do not match we can, a priori, not exclude that lines are truncated. The trick is to guarantee, as an additional condition, that the rows strictly grow in length, which guarantees that all rows are long enough to hold the complete configuration (since the length of a configuration can only grow by one symbol a time).

Hence, the domain of a grid is left- and downward-closed, and the length of the rows is strictly increasing until the last row is reached (Definition 1 and Figure 1).

Definition 1 (Grid). *Let Δ be a finite set of symbols. A* Δ-grid *is a function $g\colon D_g \mapsto \Delta$ where D_g is a finite subset of $\mathbb{N} \times \mathbb{N}$ with the following properties:*

- $\forall u, v : (u + 1, v) \in D_g \Rightarrow (u, v) \in D_g$;
- $\forall u, v : (u, v + 1) \in D_g \Rightarrow (u, v) \in D_g$;
- $\forall u, v : (u, v) \in D_g \Rightarrow (u + 1, v + 1) \in D_g$ *provided there exists w with* $(u + 1, w) \in D_g$.

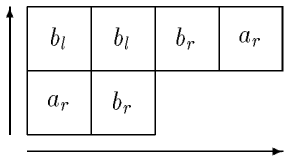
Definition 2 (Grid-term). *Let Δ be a set of symbols. A ground term t over some signature containing at least the symbols of Δ as constants, the constant $\perp$ and the ternary function symbol f, is called a* Δ-grid-term *if*

1.

$$t \in T(\Delta \cup \{\perp, f\}) \qquad \text{(PURITY)}$$

Figure 1 A grid g.

$$\begin{array}{lllll} g(0,0) & = & g(1,3) & = & a_r \\ g(1,0) & = & g(1,1) & = & b_l \\ g(0,1) & = & g(1,2) & = & b_r \end{array}$$



2. For each subterm $f(x, y, z)$:

$$x = \bot \text{ or } head(x) = f \qquad \text{(EDGE-1)}$$
$$y = \bot \text{ or } head(y) = f \qquad \text{(EDGE-2)}$$
$$z \in \Delta \qquad \text{(CONSTANTS)}$$

3. For each subterm $f(f(x_1, y_1, z_1), f(x_2, y_2, z_2), z_3)$:

$$y_1 = x_2 \qquad \text{(EQUALITY)}$$

4. For each subterm $f(f(x_1, y_1, z_1), y_2, z_2)$

$$head(y_1) = f \qquad \text{(SHAPE-1)}$$

5. For each subterm $f(x_1, f(f(x_2, y_2, z_2), y_3, z_3), z_1)$

$$head(x_1) = f \qquad \text{(SHAPE-2)}$$

Every Δ-grid-term can be seen as a grid and conversely. In fact, in a grid-term t, the last argument of an f-term is the contents 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 $\bot$ of t. The last three conditions need some explanations:

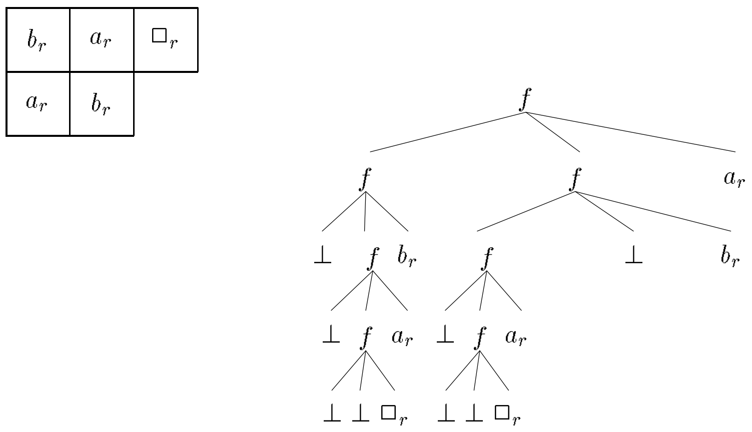
Condition EQUALITY states that by going one step up and then one step to the right one gets the same description than by first going one step to the right and then one step up. Condition SHAPE-1 states that when there is a description of the upper neighbour of some cell, then there is also a description of the upper right neighbour. Consequently, each $i+1$-th row is strictly longer than the i-th row. Finally, condition SHAPE-2 states that if a cell has an upper-right neighbour then it has an upper neighbour, too. Consequently, all rows start at the same position (see Figure 2).

The following definition makes this correspondence between grids and grid-terms precise. First we define, for any grid g, the sub-grids $\uparrow g$ and $\rightarrow g$ obtained by chopping off the first row, resp. the first column of g:

Definition 3. *Let g be a grid with a non-empty domain. The* upper *and* right sub-grids *$\uparrow g$ and $\rightarrow g$ are defined as follows:*

$$\begin{array}{rcl} D_{\uparrow g} & = & \{(n, m) \mid (n+1, m) \in D_g\} \\ \uparrow g(n, m) & = & g(n+1, m) \\ D_{\rightarrow g} & = & \{(n, m) \mid (n, m+1) \in D_g\} \\ \rightarrow g(n, m) & = & g(n, m+1) \end{array}$$

Figure 2 A grid g and its corresponding grid-term $\phi(g)$.



The translation from grids to grid-terms is now:

Definition 4. *The function ϕ, mapping Δ-grids to Δ-grid-terms, is defined as follows:*

$$\begin{array}{rcl}\phi(g) & = & \perp \quad \text{if } D_g = \emptyset \\ \phi(g) & = & f(\phi(\uparrow g), \phi(\rightarrow g), g(0,0)) \quad \text{if } g \neq \emptyset\end{array}$$

Lemma 5. *ϕ is a bijection between the set of Δ-grids and the set of Δ-grid-terms.*

Proof. Conditions PURITY, EDGE-1, EDGE-2, CONSTANTS and EQUALITY of Definition 2 are straightforward to prove. Condition SHAPE-1 corresponds to the third condition of of Definition 1 and condition SHAPE-2 corresponds to the prefix closeness condition of Δ-grid. □

Remark 6. *Since the third argument of each f is a constant c, one can encode c in f in order to obtain a binary symbols f_c. Hence the grid encoding can be done by introducing only binary symbols and constants. But proof are simpler with the ternary symbol f.*

3.2 Patterns in grids and in grid-terms

A Δ-grid-pattern is a rectangle of cells that holds symbols of Δ. More precisely:

Definition 7. *A* Δ-grid-pattern *is a function $p\colon D_p \mapsto \Delta$ where D_p is a finite subset of $\mathbb{N} \times \mathbb{N}$ of the form $\{0, \ldots, n\} \times \{0, \ldots, m\}$ (including the case of an empty domain D_p).*

Definition 8 (Matching of a grid-pattern). *A Δ-grid pattern p* matches *a Δ-grid g if there is a pair of natural numbers (n_0, m_0) such that*

- *if $(i, j) \in D_p$ then $(n_0 + i, m_0 + j) \in D_g$ and*
- *if $(i, j) \in D_p$ then $p(i, j) = g(n_0 + i, m_0 + j)$.*

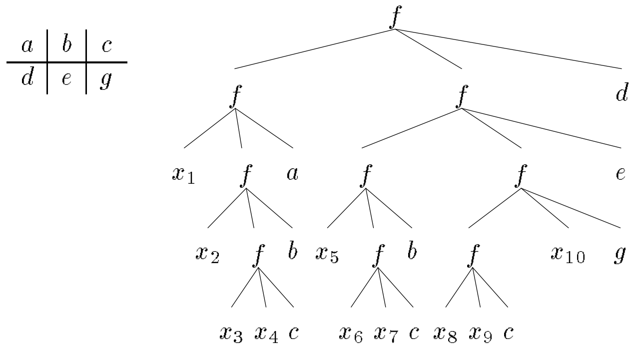
We can now translate grid-patterns to linear terms with variables. The chopping operators $\uparrow$ and $\rightarrow$ are defined on patterns analogously to grids. We use the same symbol ϕ as for the translation of grids (see Figure 3) :

Definition 9. *The translation of Δ-grid-patterns into terms is defined by*

$$\begin{aligned} \phi(p) &= \mathit{newvar} \quad \text{if } D_p = \emptyset \\ \phi(p) &= f(\phi(\uparrow p), \phi(\rightarrow p), p(0,0)) \quad \text{if } D_p \neq \emptyset \end{aligned}$$

where every occurrence of newvar stands for a fresh variable.

Figure 3 Patterns are linear terms.



We recall that a linear term π *matches* a term t if there exists a substitution σ such that $\sigma(\pi)$ is a subterm of t.

Lemma 10. *A Δ-grid-pattern p matches a Δ-grid g if and only if $\phi(p)$ matches $\phi(g)$.*

Proof. From Lemma 5. □

3.3 Turing machine computations and grids

For the rest of the paper we fix an instance of a restricted class of Turing machines. Let $T = (Q, \{a, b\}, \{a, b, \Box\}, I, q_s, \Box, \{q_f\})$ be a deterministic Turing machine where:

- $Q = \{q_1, \ldots, q_k\}$ is the set of states,
- $\{a, b\}$ is the input alphabet,
- $\{a, b, \Box\}$ is the tape alphabet,
- $I : Q \times \Gamma \to Q \times \Gamma \times \{L, 0, R\}$ is a partial transition function,
- q_s is the initial state,
- $\Box$ is the blank symbol and
- q_f is the unique final state

and such that:

- the symbol $\Box$ is never written on the tape (hence, $\Box$ can never occur in a configuration to the left of the head),
- q_s doesn't occur in the right-hand side of rules and
- q_f doesn't occur in the left-hand sides of the rules.

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 write a configuration of the Turing machine as 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, the symbols on the left half of the tape will always be indexed with l, while the symbols on the right half will be indexed with r.

Definition 11. *(TMConstants) We define the following sets of constants:*

$$\begin{aligned} \mathit{LeftChars} &:= \{a_l, b_l\} \\ \mathit{RightChars} &:= \{a_r, b_r, \Box_r\} \\ \mathit{States} &:= \{q_1, \dots, q_k\} \\ \mathit{TMConstants} &:= \mathit{LeftChars} \cup \mathit{RightChars} \cup \mathit{States} \end{aligned}$$

Hence, each instantaneous description is a word over the alphabet *TMConstants* (but not vice versa), and each computation is a finite sequence of elements from $\mathit{TMConstants}^*$.
A computation of T is successful if the first, respectively the last, instantaneous description of the computation contains q_s, respectively q_f. We use the two following well-known facts:

- The following problem is undecidable: Given a Turing machine T from the class defined above, is there a successful computation for T starting with the blank tape?

- There is a Turing machine U from the class defined above such that the following problem is undecidable: Given a natural number m, is there a successful computation of U starting with the tape consisting of $2m$-times the symbol a?

A computation sequence of a Turing machine can be arranged as a *TMConstants*-grid by writing down the instantaneous descriptions on successive rows, starting with the initial configuration on row 0 (we probably must pad $\Box_r$-symbols at the right end of rows to let the rows of the grid grow in length). Of course, not every *TMConstants*-grid represents a computation sequence. We can, however, define (Definition 12) a set of grid-patterns P_T such that a *TMConstants*-grid g represents a computation sequence of the Turing machine T iff none of the patterns of P_T matches g.
In the following, *TMConstants*-grids will simply be called *grids*, and *TMConstants*-grid-terms will be called *grid-terms*.

Definition 12 (P_T). *The set P_T is the set of patterns given in Figure 4. Here c_l, d_l denote arbitrary members of LeftChars, c_r, d_r, e_r are arbitrary members of RightChars, p, q are arbitrary members of States, and x, y, z denote arbitrary members of TMConstants.*

Note that P_T is a finite set of patterns since *TMConstants* is finite.

Lemma 13. *Let g be a grid whose first row is of the form $q_s w \Box_r \Box_r$ for some $w \in \{a_r, b_r\}^*$. Hence g represents a partial computation of the Turing machine T on the input w if and only if none of the patterns of P_T matches g.*

Figure 4 The set P_T of patterns.

$\begin{array}{c|c} \square_r & x \end{array}$ such that $x \neq \square_r$

$\begin{array}{c|c} x & q_s \end{array}$

$\begin{array}{c} q_s \\ \hline x \end{array}$

$\begin{array}{c|c} x & y \\ \hline c_l & d_l \end{array}$ such that $x \neq c_l$

$\begin{array}{c|c} x & y \\ \hline c_r & d_r \end{array}$ such that $y \neq d_r$

For every (q, d) which is not a left-hand side of any rule in I

$\begin{array}{c|c} x & y \\ \hline q & d_r \end{array}$

For every transition $(q, d) \mapsto (p, e, L)$ in I

$\begin{array}{c|c|c} x & y & z \\ \hline c_l & q & d_r \end{array}$ such that $x \neq p$ or $y \neq c_r$ or $z \neq e_r$

For every transition $(q, d) \mapsto (p, e, 0)$ in I

$\begin{array}{c|c|c} x & y & z \\ \hline c_l & q & d_r \end{array}$ such that $x \neq c_l$

$\begin{array}{c|c} x & y \\ \hline q & d_r \end{array}$ such that $x \neq p$ or $y \neq e_r$

For every transition $(q, d) \mapsto (p, e, R)$ in I

$\begin{array}{c|c|c} x & y & z \\ \hline c_l & q & d_r \end{array}$ such that $x \neq c_l$

$\begin{array}{c|c} x & y \\ \hline q & d_r \end{array}$ such that $x \neq e_l$ or $y \neq p$

Proof. If g is a grid representing a computation sequence then obviously none of the patterns of P_T applies.
Let g be a grid with the first row $q_s w \Box_r \Box_r$ such that none of the patterns of P_T applies. One can show by induction that every row is obtained from its predecessor by a computation step of the Turing machine. In the induction, we use the invariant that every instantaneous description ends on $\Box_r \Box_r$. This invariant is satisfied for the initial tape. If row i satisfies the invariant then we show that row $i+1$ (if defined), too, satisfies the invariant and is obtained from row i by one computation step. Let $c = max(\{j \mid (i,j) \in D_g\})$, i.e. c is the last column defined for row i:

- By the patterns of P_T, the symbols of row $i+1$ up to column c are obtained according to the computation rules of T.
- The symbol $\Box_r$ can only be changed to another symbol when it follows a state symbol. By the invariant this is not the case for the column c of row i, hence column c of row $i+1$ also holds $\Box_r$. By the definition of a grid the length of the rows is strictly growing, and according to the first pattern of P_T only the symbol $\Box_r$ can occur on row $i+1$ to the right of column c. Hence, the invariant holds for row $i+1$.

$\Box$

3.4 Turing machine computations and grid-terms

In the preceding sections we have seen that every computation can be coded as a grid, and that every grid can be coded as a grid-term. Hence grid-terms can be used to code computations. In Proposition 16, we characterize the set of grid-terms coding the successful computations of the Turing machine T defined at the beginning of Section 3.3.

Definition 14 (Signature Γ). *The signature Γ consists of the constants from TM-Constants, plus the ternary function symbol f and the constant $\perp$.*

The set P'_T of linear patterns is the translation of the set P_T of grid-patterns defined above:

Definition 15 (P'_T). *We define $P'_T = \{\phi(p) \mid p \in P_T\}$.*

Proposition 16. *A grid-term t on Γ codes a successful computation of the Turing machine T on the input $w = w_1 \cdots w_n$ if and only if:*

- *the term $f(x_0, f(x_1, \ldots f(x_n, f(x_{n+1}, f(x_{n+2}, \perp, \Box_r), \Box_r), w_n) \ldots), w_1), q_s)$ matches t* (INIT(w))
- *none of the patterns of P'_T matches t* (COMPUT.)
- *the term $f(\perp, x_1, q_f)$ matches t* (FINAL)

Proof. Any grid-term coding a successful computation of the Turing machine obviously fulfills the conditions of the proposition.
Let the grid-term t fulfill the conditions of the proposition. Let g be the grid such that $\phi(g) = t$. By condition INIT(w) we know that g contains on some row the sequence $q_s w \Box_r \Box_r$. Hence we conclude, from Lemma 13 and condition COMPUT. of the proposition that g encodes a computation of T. By condition FINAL, g contains the accepting state which must occur in the last row (by the restriction on the form of the Turing machine). $\Box$

4 Rewriting

4.1 Preliminaries

A finitely terminating rewrite system is said *Noetherian*. A rewrite system is said *ultra-shallow*, respectively *linear* if all its rules are ultra-shallow, respectively linear. A rewrite rule $l \to r$ is said *ultra-shallow*, respectively *linear*, if both l and r are ultra-shallow, respectively linear. A term t is *ultra-shallow* if all its variables occur at depth one (hence there is no collapsing rule of the form $l \to x$ where x is a variable in a ultra-shallow rewrite system). A term t is *linear* if it does not contain any multiple variable occurrences. An *erasing* rule rewrites a term into a smaller term w.r.t. the number of nodes. Finally, a rule of the form $f(x_1, \ldots, x_p) \to g(x_1, \ldots, x_p)$ or $a \to b$ is called a *relabeling*.

Definition 17. *The first-order language of one-step rewriting consists of one binary predicate symbol $\to$. For a given signature Σ and Σ-rewrite system R, the structure $\mathcal{A}_{\Sigma,R}$ associated with this theory is defined as follows: its universe is the set $T(\Sigma)$, and if s and t are ground terms, then $t \to_R s$ holds in $\mathcal{A}_{\Sigma,R}$ ($\mathcal{A}_{\Sigma,R} \models t \to_R s$) if and only if t rewrites to s in one rewriting step of R.*
The first-order theory of one-step rewriting of Σ and R is the set of first-order formulae over the language of one-step rewriting that are valid in $\mathcal{A}_{\Sigma,R}$.

Note that the formulae of the language on one-step rewriting do not contain the equality predicate.
In the following, we will define a rewrite system with rules and rule schemes numbered by integers. For convenience, we will write for instance "Rule (6)" for "a rule of the rule scheme (6)" and we will employ the following abbreviations in formulae:

$$x \to^1 y \text{ for } x \to y$$
$$x \to^{n+1} y \text{ for } \exists z\ (x \to z \wedge z \to^n y)$$

In proofs, we use the following notations:

$$t \underset{p}{\to} t' \text{ if } t \to t' \text{ by applying a rewrite rule at position } p \text{ in } t$$
$$t \xrightarrow{r} t' \text{ if } t \to t' \text{ by applying the rewrite rule } r$$
$$t \underset{p}{\xrightarrow{r}} t' \text{ if } t \to t' \text{ by applying the rule } r \text{ at position } p \text{ in } t$$
$$t \not\to t' \text{ if } \neg(t \to t')$$

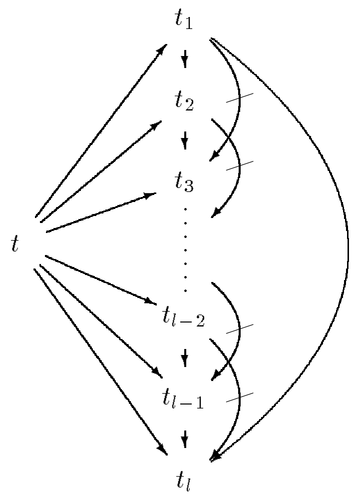
The set of leaf positions of a term t is denoted by $\mathcal{LP}os(t)$, and its set of non-leaf positions by $\mathcal{IP}os(t)$. Note that $\mathcal{P}os(t) = \mathcal{LP}os(t) \cup \mathcal{IP}os(t)$.

4.2 The Undecidability Proof

Theorem 18. *The $\exists^*\forall^*$-fragment of one-step rewriting is undecidable for any linear, ultra-shallow, Noetherian and strongly confluent rewrite system. That is there exists a linear, ultra-shallow, Noetherian and strongly confluent rewrite system for which the $\exists^*\forall^*$-fragment of one-step rewriting is undecidable.*

This theorem states that there exists a term rewriting system such that there is no algorithm which decides the satisfiability of any formula of the corresponding $\exists^*\forall^*$-fragment of one-step rewriting. To state that, we need to consider a Universal Turing machine and the associated undecidability result recalled in Section 3. In contrast, most of the previous works, except [33] prove a weaker result of the form

Figure 5 A triangle of length l.



"There is no algorithm which decides any formula of the ... -fragment of one-step rewriting with a ... term rewriting system". See [33] for further comments on that. We first give the idea of the proof.

Sketch of proof. We consider the Universal Turing machine U defined in Section 3. We define a structure $\mathcal{A}_{\Sigma_P,R_P}$ (Definitions 20 and 21) and for every natural number m a formula $halts_m$ such that $\mathcal{A}_{\Sigma_P,R_P} \models halts_m$ if and only if there is a successful computation of U starting with the tape consisting of $2m$-times the symbol a:

$$halts_m := \exists x \left(grid(x) \wedge init_m(x) \wedge trans(x) \wedge final(x) \right)$$

where

- $grid(x)$ states that a term is a Γ-grid-term, where Γ is the set of constants of U;
- $init_m(x)$ states that a term satisfies the initial condition (Condition INIT(w) of Proposition 16 with input consisting of $2m$-times the symbol a).
- $trans(x)$ states that none of the patterns of P'_T matches a term (Conditions COMPUT. of Proposition 16).
- $final(x)$ states that a term satisfies the final condition (Conditions FINAL of Proposition 16).

These formulae rely on a construction which excludes and imposes occurrence of patterns (e.g. patterns in P'_T). Let $\{\pi_1, \ldots, \pi_n\}$ be a set of patterns. The technical part of the proof is contained in the definition of a formula $match[\pi_i](x)$ (Definition 31) which states that a pattern π_i matches a term x. To this aim, we identify a pattern in a term using a sequence of rewritings called a *triangle*.
A triangle of length l is composed of $l+1$ terms $t, t_1, \ldots, t_l$ of T_{Σ_P}, satisfying the property described on Figure 5. With the rewrite system we consider, the negations $t_j \not\to t_{j+2}$ occurring in the definition of a triangle, $1 \leq j \leq l-2$, express that the rewrite chain from t_1 to t_l cannot be reduced.
The trick in the construction of $match[\pi_i](x)$ is to associate with a pattern π_i a triangle of length $i+3$ in such a way that:

one can build a triangle of length $i+3$ from x using the rewrite system R_P if and only if π_i matches x.

We "express" patterns with triangles of length greater than or equal to four, because a triangle of length 3 is used to check the purity of terms.

Definition 19 (Purity). *A term t is said* pure *if and only if $t \in T_\Gamma$.*

The idea of the proof being given, we first define the signatures Σ, Σ_P and the rewrite system R_P. In Example 22, one give an example of use of Rules (1)-(5) of R_P, and in Example 23 an example of Σ, Σ_P and R_P. Finally in Lemma 24, we prove that R_P is linear, ultra-shallow and Noetherian.

Definition 20 (Signatures Σ, Σ_P). *The signature Σ is the extension of Γ by two ternary symbols u and r, one unary symbol w and four constants A, B, C, D. Let $P = \{\pi_1, \ldots, \pi_n\}$ be a set of terms of $T_\Sigma(\mathcal{X})$. The signature Σ_P is defined as follows:*

$$\begin{aligned} \Sigma_P \quad := \quad & \Sigma \\ \cup \; & \{c_{\pi,o}/0 \mid \pi \in P, o \in \mathcal{IP}os(\pi)\} \\ \cup \; & \{e_{i,j}/0 \mid 1 \le i \le n, 1 \le j \le i+3\} \\ \cup \; & \{\overline{f}/3, \tilde{f}/3, \tilde{u}/3, \tilde{r}/3, \tilde{w}/1\} \end{aligned}$$

Definition 21 (Rewrite system R_P). *The rewrite system R_P consists of the following rules:*

- *Rules (1)-(5) are used to check whether the code of the initial configuration matches a term (Condition* Init*(w) of Proposition 16):*

$$\{f(x,y,z) \to \overline{f}(x,y,z)\} \tag{1}$$
$$\{\overline{f}(x,\perp,\Box_r) \to A\} \tag{2}$$
$$\{f(x,A,\Box_r) \to B\} \tag{3}$$
$$\{f(x,B,a_r) \to C, f(x,C,a_r) \to B\} \tag{4}$$
$$\{f(x,B,q_s) \to D\} \tag{5}$$

- *Rules (6)-(8) are used to check that a term satisfies characteristic equalities of grid-terms (Condition* Equality *of Definition 2):*

$$\{f(x,y,z) \to u(x,y,z), f(x,y,z) \to r(x,y,z)\} \tag{6}$$
$$\{u(x,y,z) \to w(x)\} \tag{7}$$
$$\{r(x,y,z) \to w(y)\} \tag{8}$$

- *Rules (9)-(11) construct a triangle of length 3 in order to check (non-)purity of terms:*

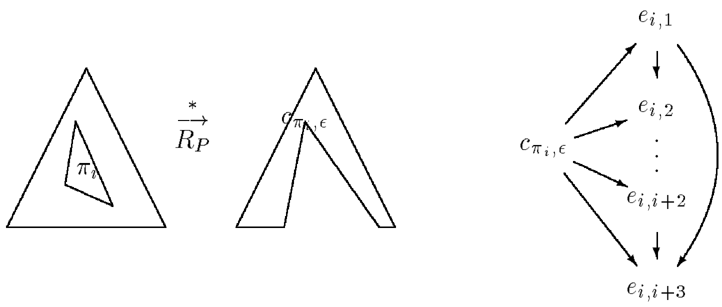
$$\{a_r \to b_r, b_r \to \Box_r, a_r \to \Box_r\} \tag{9}$$
$$\{\alpha \to a_r, \alpha \to b_r, \alpha \to \Box_r \mid \alpha \in \Sigma_P \backslash \Gamma, arity(\alpha) = 0\} \tag{10}$$
$$\begin{aligned}\{\beta(x_1,\ldots,x_p) \to a_r, \beta(x_1,\ldots,x_p) \to b_r, \beta(x_1,\ldots,x_p) \to \Box_r \mid \\ \beta \in \Sigma_P \backslash \Gamma, arity(\beta) = p, p > 0\}\end{aligned} \tag{11}$$

- *Rules (12)-(13) reduce a pattern (Figure 6(a)) to a special constant:*

$$\{h(x_1,\ldots,x_p) \to \tilde{h}(x_1,\ldots,x_p) \mid h = f, r, u \text{ or } w\} \tag{12}$$
$$\begin{aligned}\{\tilde{h}(d_{\pi,o1},\ldots,d_{\pi,op}) \to c_{\pi,o} \mid \\ \pi \in P, o \in \mathcal{IP}os(\pi), head(\pi|_o) = h, arity(h) = p\}\end{aligned} \tag{13}$$

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

Figure 6 How to use R_P to check whether a pattern matches a term.



(a) The reduction of a pattern using rules 12-13 of R_P. In the left part, π_i matches a term; in the right part the pattern is replaced by a special constant $c_{\pi_i,\epsilon}$.

(b) Identification of a pattern π_i using a triangle of length $i+3$.

- *Rules (14)-(16) construct a triangle (Figure 6(b)) in order to identify a pattern:*

$$\{c_{\pi_i,\epsilon} \to e_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq i+3\} \tag{14}$$
$$\{e_{i,j} \to e_{i,j+1} \mid 1 \leq i \leq n, 1 \leq j \leq i+2\} \tag{15}$$
$$\{e_{i,1} \to e_{i,i+3} \mid 1 \leq i \leq n\} \tag{16}$$

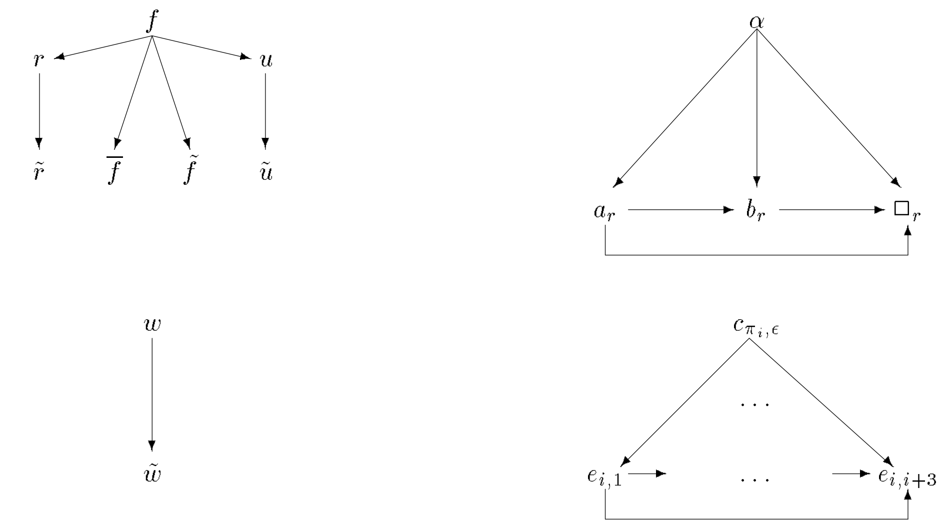
Let us remark that:

- The rewrite system R_P consists of relabelings ((1), (6), (9), (10), (12), (14), (15), (16)) and of erasing rules ((2), (3), (4), (5), (7), (8), (11), (13)). A schema of the relabelings of R_P is given in figure 7.
- Only rules (1), (6), (9) and (12) apply to a pure term. In particular, no erasing rule applies to a pure term.
- Every term of T_{Σ_P} except $a_l, b_l, q_1, \ldots, q_k$ reduces by R_P to $\Box_r$.
- For every terms t_1 and t_2, if $t_1 \xrightarrow{p} t_2$ then $t_1(p) \neq t_2(p)$.

Example 22. *Let m be an integer, $t_0, \ldots, t_{2m+2}$ be terms of T_Γ, S be a context of $T_\Gamma(\mathcal{X})$ and t be the term*

$$S\big[f(t_0, f(t_1, \ldots f(t_{2m}, f(t_{2m+1}, f(t_{2m+2}, \bot, \Box_r), \Box_r), a_r) \ldots), a_r), q_s)\big]$$

Figure 7 Schema of the relabeling rules.



containing $2m$ *occurrences of the symbol* a_r. *One can do the following rewritings:*

$$
\begin{array}{c}
S\big[f(t_0, f(t_1, \ldots f(t_{2m}, f(t_{2m+1}, f(t_{2m+2}, \bot, \Box_r), \Box_r), a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (1)} \\
S\big[f(t_0, f(t_1, \ldots f(t_{2m}, f(t_{2m+1}, \overline{f}(t_{2m+2}, \bot, \Box_r), \Box_r), a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (2)} \\
S\big[f(t_0, f(t_1, \ldots f(t_{2m}, f(t_{2m+1}, A, \Box_r), a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (3)} \\
S\big[f(t_0, f(t_1, \ldots f(t_{2m}, B, a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (4)} \\
S\big[f(t_0, f(t_1, \ldots f(t_{2m-1}, C, a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (4)} \\
S\big[f(t_0, f(t_1, \ldots f(t_{2m-2}, B, a_r) \ldots), a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (4)} \\
\ldots \\
\Big\downarrow{\scriptstyle (4)} \\
S\big[f(t_0, f(t_1, C, a_r), q_s)\big] \\
\Big\downarrow{\scriptstyle (4)} \\
S\big[f(t'_0, B, q_s)\big] \xrightarrow{(5)} S[D]
\end{array}
$$

Example 23. *If* $\pi_1 = f(x, f(y, z, a_r), \Box)$ *and* $P = \{\pi_1\}$*, then*

$$\begin{array}{rcl} \Sigma_P & := & \Sigma \\ & & \cup\ \{c_{\pi_{1,\epsilon}}/0, c_{\pi_{1,2}}/0\} \\ & & \cup\ \{e_{1,1}/0, e_{1,2}/0, e_{1,3}/0, e_{1,4}/0\} \\ & & \cup\ \{\overline{f}/3, \tilde{f}/3, \tilde{u}/3, \tilde{r}/3, \tilde{w}/1\} \end{array}$$

and Rules (13)-(16) are the following

$$\begin{array}{lr} \{\tilde{f}(y, z, a_r) \to c_{\pi_{1,2}}, \tilde{f}(x, c_{\pi_{1,2}}, \Box) \to c_{\pi_{1,\epsilon}}\} & (13) \\ \{c_{\pi_1,\epsilon} \to e_{1,1}, c_{\pi_1,\epsilon} \to e_{1,2}, c_{\pi_1,\epsilon} \to e_{1,3}, c_{\pi_1,\epsilon} \to e_{1,4}\} & (14) \\ \{e_{1,1} \to e_{1,2}, e_{1,2} \to e_{1,3}, e_{1,3} \to e_{1,4}\} & (15) \\ \{e_{1,1} \to e_{1,4}\} & (16) \end{array}$$

For all terms $t_1, t_2, t_3 \in T_\Gamma$*, we can write the following derivations:*
$f(t_1, f(t_2, t_3, a_r), \Box) \xrightarrow{(12)^*} \tilde{f}(t_1, \tilde{f}(t_2, t_3, a_r), \Box) \xrightarrow{(13)^*} f(t_1, c_{\pi_{1,2}}, \Box) \xrightarrow{(13)^*} c_{\pi_{1,\epsilon}}$ *and*

$$\begin{tikzcd} & e_{1,1} \arrow[d] \arrow[ddd, bend left] \\ & e_{1,2} \arrow[d] \\ c_{\pi_1,\epsilon} \arrow[uur] \arrow[ur] \arrow[r] \arrow[dr] & e_{1,3} \arrow[d] \\ & e_{1,4} \end{tikzcd}$$

Lemma 24. *The rewrite system* R_P *is linear, ultra-shallow, Noetherian and strongly confluent.*

Proof. The rewrite system R_P is obviously linear and ultra-shallow. We prove that R_P is Noetherian with a lexicographic path ordering over the set of terms T_{Σ_P} defined from the ordering $\succ$ over Σ_P:

$$\begin{array}{rl} \forall \alpha \in \{A, B, C, D\}, & f \succ \alpha \\ & f \succ \overline{f} \succ u \succ r \succ w \\ \forall \alpha \in \Sigma_P \backslash \Gamma, & \alpha \succ a_r \succ b_r \succ \Box_r \\ \forall h \in \{f, u, r, w\}, \forall \pi \in P, \forall o \in \mathcal{IP}os(\pi), & h \succ \tilde{h} \succ c_{\pi,o} \\ \forall i \in \{1, \ldots, n\}, & c_{\pi_i,\epsilon} \succ e_{i,1} \succ e_{i,2} \succ \cdots \succ e_{i,l+i} \end{array}$$

Let $\succ_{lpo}$ be the lexicographic path order on T_{Σ_P} induced by $\succ$. For any rule $l \to r$ of the rewrite system R_P, we can prove that $l \succ_{lpo} r$. We deduce that the rewrite system R_P is Noetherian.
Finally we can prove that R_P is strongly confluent since all its critical pairs are joignable in one rewrite step. In fact for each critical pair $\langle s, t\rangle$, $s \to \Box_r$ and $t \to \Box_r$ since $s, t \in T_{\Sigma_P} \setminus \{a_l, b_l, q_1, \ldots, q_k\}$. $\Box$

We now give four technical properties of the rewrite system R_P (Propositions 25, 26, 27 and 28). These properties are used in the proof of Theorem 18.

Proposition 25. *Let* $t, t_1, t_2 \in T_{\Sigma_P}$ *satisfy the property described on Figure 8(a). If* $t \xrightarrow{(r)} t_2$ *and* r *is a relabeling then* $p_1 = p_2 = p_3$ *and only relabelings are used in the diagram.*

Figure 8 Four technical properties on R_P

$$\begin{tikzcd} t \arrow[r, "p_1"'] \arrow[dr, "p_2"'] & t_1 \arrow[d, "p_3"'] \\ & t_2 \end{tikzcd}$$

(a) Propositions 25, 26 and 27

$$\begin{tikzcd} & t_1 \arrow[d] \arrow[dd, bend left=60, "/" marking] \\ t \arrow[ur] \arrow[r] \arrow[dr] & t_2 \arrow[d] \\ & t_3 \end{tikzcd}$$

(b) Proposition 28

Proof. Since r is a relabeling and R_P contains only erasing rules and relabelings, the number of nodes of t, t_1, and t_2 are equal. Hence, only relabelings are used in rewritings of Figure 8(a).
Moreover if $p_1 \neq p_3$ then t and t_2 differ at two distinct positions, hence $t \not\to t_2$ by a relabeling of R_P. Similarly $t_1 \not\to t_2$ if $p_1 \neq p_2$ since in this case t_1 and t_2 differ at two distinct positions. We deduce that $p_1 = p_2 = p_3$. □

Proposition 26. *Let $t, t_1, t_2 \in T(\Sigma_P)$ satisfy the property described on Figure 8(a) and such that at least one erasing rule is applied in the rewritings. Therefore $t \to t_2$ by an erasing rule and $p_2 \trianglelefteq p_1$.*

Proof. Let us consider $t, t_1, t_2 \in T(\Sigma_P)$ satisfying the property described on Figure 8(a) and such that at least one erasing rule is applied in the rewritings. According to the number of positions of t and t_2, we can prove that $t \to t_2$ by an erasing rule.
Let us now prove that $p_2 \trianglelefteq p_1$. We distinguish two cases.

First case: p_1 and p_2 are parallel. Therefore t_1 and t_2 differ in two incomparable positions. Since $t_1 \to t_2$, we must have $p_3 \triangleleft p_1$ and $p_3 \triangleleft p_2$.

$$\begin{aligned} t_1 \underset{p_3}{\longrightarrow} t_2 &\Rightarrow t_1(p_3) \neq t_2(p_3) \\ p_3 \triangleleft p_1 \text{ and } p_3 \triangleleft p_2 &\Rightarrow t_1(p_3) = t_2(p_3) \end{aligned}$$

Therefore we obtain a contradiction.

Second case: $p_1 \triangleleft p_2$. For all positions p such that p and p_1 are parallel or $p \triangleleft p_1$, we have $t_1(p) = t_2(p)$. Moreover $t_1(p_1) \neq t_2(p_1)$ since $t_2(p_1) = t(p_1)$ and $t \underset{p_1}{\longrightarrow} t_1$. We deduce that $p_3 = p_1$ according to the rules of R_P. Therefore there exist two rules $l_1 \to r_1$ and $l_2 \to r_2$ of R_P such that :

$$t \xrightarrow{l_1 \to r_1} t_1 \xrightarrow{l_2 \to r_2} t_2$$

with $head(r_1) = head(l_2)$ et $head(r_2) = head(l_1)$. But there aren't such rules in R_P hence $p_1 \triangleleft p_2$ is impossible.

Hence $p_2 \trianglelefteq p_1$ must hold. □

Proposition 27. *Let $t, t_1, t_2 \in T(\Sigma_P)$ satisfy the property described on Figure 8(a). If $t \to t_2$ by the rule (7) or (8), then the rewriting $t \to t_1$ is done on a position erased by the rewriting $t \to t_2$. That is, if $t \xrightarrow{(7)} t_2$ (respectively $t \xrightarrow{(8)} t_2$), then there exists α equal to 2 or 3 (respectively 1 or 3), and p such that $p_1 = p_2 \alpha p$.*

Proof. Let us consider $t, t_1, t_2 \in T(\Sigma_P)$ satisfying the property described on Figure 8(a) and let us suppose that $t \xrightarrow{(7)} t_2$. There exist $s_1, s_2, s_3 \in T_{\Sigma_P}$ such that $t|_{p_2} = u(s_1, s_2, s_3)$.
Rule (7) is erasing hence $p_2 \trianglelefteq p_1$. Let us first suppose that $p_2 = p_1$. Therefore $t(p_1) = u$. Let us denote by r_1 the rule such that $t \xrightarrow[p_1]{r_1} t_1$, Therefore either $r_1 = (7)$, or $r_1 = (11)$, or $r_1 = (12)$. In the first case, $t_1 = t_2$. In the second case, the number of positions of t_1 is strictly smaller than the number of positions of t_2. And in the last case, $t_1|_{p_1} = \tilde{u}(s_1, s_2, s_3)$ and $t_2|_{p_1} = w(s_1)$. In all three cases $t_1 \not\to t_2$. We deduce that $p_2 \triangleleft p_1$.
There exists $\alpha \in \{1, 2, 3\}$ and p such that $p_1 = p_2 \alpha p$. Let us suppose that $\alpha = 1$. Therefore there exists $s'_1 \in T_{\Sigma_P}$, $s'_1 \neq s_1$ such that $t_1|_{p_2} = u(s'_1, s_2, s_3)$. We deduce that $t_1 \not\to t_2$ since $t_2|_{p_2} = w(s_1)$. Finally $\alpha = 2$ or 3. The proof for rule (8) is analogous. □

Proposition 28. *For all $t, t_1, t_2, t_3 \in T(\Sigma_P)$ satisfying the property described on Figure 8(b), no erasing rule is applied in the different rewritings.*

Proof. Let $t, t_1, t_2, t_3 \in T(\Sigma_P)$ satisfying the property described on Figure 8(b). Let us consider that $\forall i \in \{1, 2, 3\}$, $t \xrightarrow[p_i]{r_i} t_i$.
Let us suppose that an erasing rule is applied. According to Propositions 25 and 26, r_3 is an erasing rule and $p_3 \trianglelefteq p_2 \trianglelefteq p_1$. Hence r_3 is one of the following rules: (2), (3), (4), (5), (7), (8), (11) or (13).
We distinguish the cases where r_3 is rule (2), rule (3), rule (7) and rule (11) in order to prove that either $t_1 \to t_3$, or there is no term t_2 such that $t_1 \to t_2 \to t_3$. The case where r_3 is rule (4) or (5) is similar to the case of rule (3), and the case where r_3 is rule (8), respectively rule (13), is similar to the case of rule (7), respectively (11).

Case where r_3 is rule (2). There exists $s \in T_{\Sigma_P}$ such that $t|_{p_3} = \overline{f}(s, \bot, \Box_r)$ and $t_3|_{p_3} = A$.

If $p_3 \triangleleft p_1$, then $t_1 \xrightarrow[p_3]{(2)} t_3$. If $p_3 = p_1$, then either $t \xrightarrow[p_3]{(2)} t_1$, or $t \xrightarrow[p_3]{(11)} t_1$. Hence there is no term t_2 such that $t_1 \to t_2 \to t_3$.

Case where r_3 is rule (3). There exists $s \in T_{\Sigma_P}$ such that $t|_{p_3} = f(s, A, \Box_r)$ and $t_3|_{p_3} = B$.

If $p_3 \triangleleft p_1$, then either $\exists s' \in T_{\Sigma_P}$ such that $t_1|_{p_3} = f(s', A, \Box_r)$, or $\exists \alpha \in \{a_r, b_r, \Box_r\}$ such that $t_1|_{p_3} = f(s, \alpha, \Box_r)$. In the first case, $t_1 \xrightarrow[p_3]{(3)} t_3$ and in the second case, $\nexists t_2$ such that $t_1 \to t_2 \to t_3$.

If $p_3 = p_1$, then either $t \xrightarrow[p_3]{(3)} t_1$, or $t \xrightarrow[p_3]{(11)} t_1$. Hence $t_1 \to t_2 \to t_3$ is impossible.

Case where r_3 is rule (7). According to Proposition 27, there exists α equal to 2 or 3 and p such that $p_2 = p_3 \alpha p$. Hence $t_1 \xrightarrow[p_3]{(7)} t_3$ since $p_2 \trianglelefteq p_1$.

Case where r_3 is rule (11). There exists $\beta \in \Sigma_P \setminus \Gamma$, $s_1, \ldots, s_p \in T_{\Sigma_P}$ (p arity of β) and $c \in \{a_r, b_r, \Box_r\}$ such that $t|_{p_3} = \beta(s_1, \ldots, s_p)$ and $t_3|_{p_3} = c$.

If $p_3 \triangleleft p_1$, then $t_1 \xrightarrow[p_3]{(11)} t_3$. If $p_3 = p_1$, then since $t_1 \not\to t_3$, $t_1(p_3)$ belongs to Γ otherwise $t_1 \xrightarrow[p_3]{} t_3$ by the rule 10 or 11. Hence $t \xrightarrow[p_3]{(11)} t_1$. Therefore

$t_1(p_3)$ and $t_3(p_3)$ belong to $\{a_r, b_r, \Box_r\}$. Since $t_1 \to t_2 \to t_3$, we deduce that $t_1(p_3) = a_r$, $t_2(p_3) = b_r$ and $t_3(p_3) = \Box_r$ and hence $t_1 \xrightarrow[p_3]{(9)} t_3$.

In all cases, either $t_1 \to t_3$ or there is no term t_2 such that $t_1 \to t_2 \to t_3$ which contradicts hypothesis of the Figure 8(b). We deduce that no erasing rule is applied in the rewriting of Figure 8(b). $\Box$

Let us now define the formula $impure(x)$ (Definition 29) and prove that a term t satisfies Condition Purity of Definition 2 if and only if $\mathcal{A}_{\Sigma_P,R_P} \models \neg impure(t)$ (Lemma 30).

Definition 29 **(**$impure(x)$**).**

$$impure(x) := \exists y_1, y_2, y_3 \quad \begin{tikzcd} & y_1 \arrow[d] \arrow[dd, bend left] \\ x \arrow[ur] \arrow[r] \arrow[dr] & y_2 \arrow[d] \\ & y_3 \end{tikzcd}$$

Lemma 30. *For every $t \in T_{\Sigma_P}$: $\mathcal{A}_{\Sigma_P,R_P} \models \neg impure(t)$ if and only if t is pure.*

Proof. The $\Rightarrow$ direction is obvious since for every $t \in T_{\Sigma_P}$, if $t \notin T_\Gamma$ then $\mathcal{A}_{\Sigma_P,R_P} \models impure(t)$ according to rules (10) and (11). We only prove the direction $\Leftarrow$.
Let t be a term of T_Γ. Let us suppose that there exist $t_1, t_2, t_3 \in T_{\Sigma_P}$ satisfying:

$$\begin{tikzcd} & t_1 \arrow[d] \\ t \arrow[ur, "p_1"'] \arrow[r, "p_2"'] \arrow[dr, "p_3"] & t_2 \arrow[d] \\ & t_3 \end{tikzcd}$$

Erasing rules don't apply to t since t is a pure term. Hence, according to Proposition 25, $p_1 = p_2 = p_3$ and there exist a context C, symbols $\alpha \in \Gamma$, $\beta, \gamma, \delta \in \Sigma_P$, all of the same arity p, and terms $s_1, \ldots, s_p \in T_\Gamma$ such that:

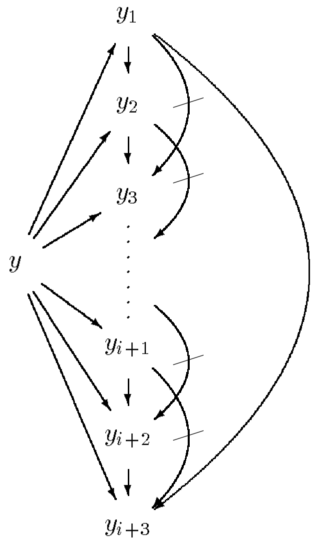
$$\begin{array}{rclcrcl} t & = & C(\alpha(s_1, \ldots, s_p)) & \quad & t_1 & = & C(\beta(s_1, \ldots, s_p)) \\ t_2 & = & C(\gamma(s_1, \ldots, s_p)) & & t_3 & = & C(\delta(s_1, \ldots, s_p)) \end{array}$$

Moreover only rules (1), (6), (9) and (12) apply to t since t is pure. Therefore symbols β, γ, δ belong to the set $\{u, r, b_r, \Box_r, \overline{f}, \tilde{f}\}$. Hence by inspection of the rewriting rules, $t_1 \to t_2 \to t_3$ is impossible which contradicts the hypothesis. $\Box$

Let us now define the formula $match[\pi](x)$ (Definition 31) and prove that a pattern π matches a term t if and only if $\mathcal{A}_{\Sigma_P,R_P} \models match[\pi](t)$ (Lemma 33).

Definition 31 **(**$match[\pi](x)$**).** *For every i, $1 \le i \le n$, let k_i be the cardinality of $\mathcal{IP}os(\pi_i)$ and:*

$match[\pi_i](x) := \exists y, y_1, \ldots, y_{i+3}\ x \rightarrow^{2k_i}$



The first part of the formula $x \rightarrow^{2k_i} y$ reduces a pattern π_i to a constant $c_{\pi_i,\epsilon}$. The second part forms a triangle used to identify via its length the matching pattern.

Remark 32. *If $k_i = 0$, then $x = y$.*

Lemma 33. *For every $i \in \{1, \ldots, n\}$ and $t \in T_\Sigma$:*
$\mathcal{A}_{\Sigma_P,R_P} \models match[\pi_i](t)$ *if and only if π_i matches t.*

Proof. Let $i \in \{1, \ldots, n\}$ and $t \in T_\Sigma$. Let us suppose that π_i matches t. Therefore there exists a linear term C of $T_\Sigma(\mathcal{X})$ and m terms $t_1, \ldots, t_m$ of T_Σ such that $t = C\big[\pi_i[t_1, \ldots, t_m]\big]$.

$\forall o \in \mathcal{IP}os(\pi_i)$ such that $head(\pi_i|_o) = h$ and $h \in \Sigma_p$, we have $h(d_{\pi_i,o1}, \ldots, d_{\pi_i,op}) \xrightarrow{(12)} \tilde{h}(d_{\pi_i,o1}, \ldots, d_{\pi_i,op}) \xrightarrow{(13)} c_{\pi_i,o}$ since $p > 0$. We deduce $t \rightarrow^{2k_i} C[c_{\pi_i,\epsilon}]$ and $\mathcal{A}_{\Sigma_P,R_P} \models match[\pi_i](t)$.

Let us now suppose that $\mathcal{A}_{\Sigma_P,R_P} \models match[\pi_i](t)$. According to Definition 31, there exist $t', t_1, \ldots, t_{i+3} \in T_{\Sigma_P}$ such that $t \rightarrow^{2k_i} t'$ and $t', t_1, \ldots, t_{i+3}$ satisfy the property described on Figure 9.
First let us prove that $c_{\pi_i,\epsilon}$ occurs in t'. For all j, $1 \leq j \leq i+3$, let us suppose that $t' \rightarrow t_j$ at position p_j in t. According to Proposition 28 no erasing rules is applied and according to Proposition 25, all positions p_j are equal to a position denoted p. Thus only relabelings are done along the sequence depicted in Figure 9 that is only rules (1), (6), (9), (10), (12), (14), (15) and (16) are used in these rewritings. Let α be the symbol at position p in t'. Let us remark that:

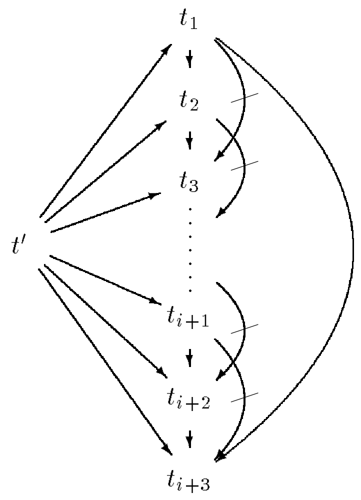
- Rules (1), (6) and (12) are the only rules which can be applied at a non-leaf position. Therefore for all j, $1 \leq j \leq i+3$, $t_j(p) \in \{\overline{f}, \tilde{f}, u, r, \tilde{u}, \tilde{r}\}$. We deduce that it is impossible to construct a triangle if p is a non-leaf position.

- If $\alpha \in \Gamma$, then only rule (9) is applied and gives triangles of length at most 3.

- A triangle with at least one application of rule (10) can be reduced using rule (9) or (10). This contradicts the formula $\bigwedge_{1 \leq j \leq i+1} t_j \not\rightarrow t_{j+2}$.

  In fact, let us suppose rule (10) is applied. Let k be the index of the term t_j obtained by the first application of this rule. Therefore $t_k|_p \in \{a_r, b_r, \Box_r\}$.

Figure 9



- If $k = 1$ or 2, then $t' \xrightarrow{(10)} t_{k+1}$ and $t' \xrightarrow{(10)} t_{k+2}$ since $t' \underset{p}{\rightarrow} t_{k+1}$, $t' \underset{p}{\rightarrow} t_{k+2}$ and $t_k \rightarrow t_{k+1} \rightarrow t_{k+2}$. Hence $t_k|_p = a_r$, $t_{k+1}|_p = b_r$ and $t_{k+2}|_p = \Box_r$. Finally $t_k \underset{p}{\xrightarrow{(9)}} t_{k+2}$.
- If $k > 2$, then according to the previous points, $t' \rightarrow t_{k-2}$ by application of the rule (14), (15) or (16). Therefore there exists $l \in \{1, \ldots, n\}$ and $m \in \{1, \ldots, l{+}3\}$ such that $t_{k-2}(p) = e_{l,m}$. We deduce that $t_{k-2} \underset{p}{\xrightarrow{(10)}} t_k$.

- Since R_P is Noetherian, all t_j are pairwise different. Moreover, among the rules (14), (15), (16), only rule (14) can be applied to t' in order to lead to more than two different terms.

Consequently, one of the symbols $\{c_{\pi_j,\epsilon} \mid 1 \leq j \leq n\}$ occurs in t' and t' is rewritten $i{+}3$ times using rule (14). Moreover, each one of the t_j's is rewritten using rules (15) and (16) since $t_j \rightarrow t_{j+1}$. Clearly $t_1 \xrightarrow{(16)} t_{i+3}$. Consequently for each j, $e_{i,j}$ occurs in t_j and $c_{\pi_i,\epsilon}$ occurs in t'.

It remains to prove that if $t \rightarrow^{2k_i} t'$ and $c_{\pi_i,\epsilon}$ occurs in t', then π_i matches t. Since $t \in T_\Sigma$, the only possibility to obtain one occurrence of $c_{\pi_i,\epsilon}$ in t' is to use k_i applications of both rules (12) and (13). Finally π matches t if $\mathcal{A}_{\Sigma_P,R_P} \models match[\pi](t)$.

$\Box$

Let us now define the formula $equal(x)$ (Definition 34) and prove that any pure term t satisfies Condition Equality of Definition 2 if and only if $\mathcal{A}_{\Sigma_P,R_P} \models equal(t)$.

Definition 34 ($equal(x)$).

$$
\begin{aligned}
equal(x) :=\ & \forall y, z, z', s, s', y' \\
& \Big(x \to^2 y \wedge match[f(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7)](y) \\
& \quad \wedge y \to z \wedge match[u(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7)](z) \\
& \quad \wedge z \to s \wedge match[u(w(x_1),u(x_2,x_3,x_4),x_5)](s) \\
& \quad \wedge s \to y' \wedge match[w(w(x_1))](y') \\
& \quad \wedge y \to z' \wedge match[r(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7](z') \\
& \quad \wedge z' \to s' \wedge match[r(r(x_1,x_2,x_3),w(x_4),x_5)](s')\Big) \;\;\Rightarrow\;\; s' \to y'
\end{aligned}
$$

Lemma 35. *For every $t \in T_{\Sigma_P}$:*

$$\mathcal{A}_{\Sigma_P,R_P} \models \neg impure(t) \wedge equal(t)$$
if and only if
t is pure and satisfies Condition Equality *of Definition 2.*

Proof. Let us consider the $\Rightarrow$ direction (the other direction being straightforward). Let t be a term of T_{Σ_P} such that $\mathcal{A}_{\Sigma_P,R_P} \models \neg impure(t) \wedge equal(t)$. According to Lemma 30, t is pure.
Let us now suppose there exists $t_1,\ldots,t_7 \in T_\Gamma$ such that $f(f(s_1,s_2,s_3),f(s_4,s_5,s_6),s_7)$ is a subterm of t. Therefore there exists a context C such that

$$t = C(f(f(s_1,s_2,s_3),f(s_4,s_5,s_6),s_7)).$$

We have the following properties:

$$
\begin{array}{lcl}
t \xrightarrow{(6)} \xrightarrow{(6)} t_1 & \wedge & match[f(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7)](t_1) \\
 & & \quad\text{with } t_1 = C\big[f(r(s_1,s_2,s_3),u(s_4,s_5,s_6),s_7)\big] \\
t_1 \xrightarrow{(6)} t_2 & \wedge & match[u(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7)](t_2) \\
 & & \quad\text{with } t_2 = C\big[u(r(s_1,s_2,s_3),u(s_4,s_5,s_6),s_7)\big] \\
t_2 \xrightarrow{(8)} t_3 & \wedge & match[u(w(x_2),u(x_2,x_3,x_4),x_5)](t_3) \\
 & & \quad\text{with } t_3 = C\big[u(w(s_2),u(s_4,s_5,s_6),s_7)\big] \\
t_3 \xrightarrow{(7)} t_4 & \wedge & match[w(w(x_2))](t_4) \\
 & & \quad\text{with } t_4 = C\big[w(w(s_2))\big] \\
t_1 \xrightarrow{(6)} t_5 & \wedge & match[r(r(x_1,x_2,x_3),u(x_4,x_5,x_6),x_7)](t_5) \\
 & & \quad\text{with } t_5 = C\big[r(r(s_1,s_2,s_3),u(s_4,s_5,s_6),s_7)\big] \\
t_5 \xrightarrow{(7)} t_6 & \wedge & match[r(r(x_1,x_2,x_3),w(x_4),x_5)](t_6) \\
 & & \quad\text{with } t_6 = C\big[r(r(s_1,s_2,s_3),w(s_4),s_7)\big]
\end{array}
$$

Since $\mathcal{A}_{\Sigma_P,R_P} \models equal(t)$, we deduce that $t_6 \to t_4$. The only rule one can apply is rule (8). Hence $t_4 = C(w(w(s_2))) = C(w(w(s_4)))$ and $s_2{=}s_4$. Hence, t satisfies Condition Equality of Definition 2. ☐

Let us now define the formula $grid(x)$ (Definition 36) and prove that a term is a Γ-grid-term if and only if $\mathcal{A}_{\Sigma_{P'},R_{P'}} \models grid(t)$ where P' is the set of patterns occurring in the definition of the formula $grid(x)$.

Definition 36 ($grid(x)$)**.**

$$
\begin{array}{lll}
grid(x) := & & \\
& \neg impure(x) & \text{(R-PURE)} \\
\wedge & \bigwedge_{h\in\Gamma_0\setminus\{\bot\}} \Big(\neg match[f(h,x_1,x_2)](x) \ \wedge\ \neg match[f(x_1,h,x_2)](x)\Big) & \\
& & \text{(R-EDGES1\&2)} \\
\wedge & \Big(\neg match[f(x_1,x_2,\bot)](x) \ \wedge\ \neg match[f(x_1,x_2,f(x_3,x_4,x_5))](x)\Big) & \\
& & \text{(R-CONSTANTS)} \\
\wedge & equal(x) & \text{(R-EQUAL)} \\
\wedge & \bigwedge_{h\in\Gamma_0} \neg match[f(f(x_1,h,x_2),x_3,x_4)](x) & \text{(R-SHAPE1)} \\
\wedge & \bigwedge_{h\in\Gamma_0} \neg match[f(h,f(f(x_1,x_2,x_3),x_4,x_5),x_6)](x) & \text{(R-SHAPE2)}
\end{array}
$$

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

Lemma 37. *Let $\Sigma_{P'} \supseteq \Sigma$ and $R_{P'} \supseteq R$ be constructed according to Definitions 20 and 21, where P' contains all the patterns mentioned in Definition 36. Therefore a term $t \in T(\Sigma_{P'})$ is a Γ-grid-term if and only if $\mathcal{A}_{\Sigma_{P'},R_{P'}} \models grid(t)$.*

Proof. The correspondence between conditions of Definition 2 and formulae of Definition 36 is obvious. $\square$

Let us now define for every natural number m the formula $init_m$ which is used to check that a term satisfies Condition $\textsc{Init}(w)$ of Proposition 16 with input w consisting of $2m$-times the symbol a.

Definition 38 ($init_m(x)$)**.** *For every natural number m, let:*

$$
\begin{array}{llll}
init_m(x) \ := \ \exists y_1, y_2, z_1, \dots, z_{2m+1}, s & & & \\
\quad x \to y_1 & \wedge\ match[\overline{f}(x_1,\bot,\square_r)](y_1) & & \\
\quad \wedge\ y_1 \to y_2 & \wedge\ match[A](y_2) & \wedge\ y_2 \to z_1 & \wedge\ match[B](z_1) \\
\quad \wedge\ z_1 \to z_2 & \wedge\ match[C](z_2) & \wedge\ z_2 \to z_3 & \wedge\ match[B](z_3) \\
\quad \dots & & & \\
\quad \wedge\ z_{2m-1} \to z_{2m} & \wedge\ match[C](z_{2m}) & \wedge\ z_{2m} \to z_{2m+1} & \wedge\ match[B](z_{2m+1}) \\
\quad \wedge\ z_{2m+1} \to s & \wedge\ match[D](s) & &
\end{array}
$$

Note that for every natural number m, $init_m(x)$ is a $\exists^*$ formula since each occurrence of a *match*-formula is positive.

Lemma 39. *Let $\Sigma_{P'} \supseteq \Sigma$ and $R_{P'} \supseteq R$ be constructed according to Definitions 20 and 21, where P' contains all the patterns mentioned in Definition 38. For every $m \in \mathbb{N}$, the pattern*

$$f(x_0, f(x_1, \dots f(x_{2m}, f(x_{2m+1}, f(x_{2m+2}, \bot, \square_r), \square_r), a_r) \dots), a_r), q_s)$$

containing $2m$ occurrences of a_r matches a term $t \in T_\Gamma$ if and only if $\mathcal{A}_{\Sigma_{P'},R_{P'}} \models init_m(t)$.

Proof. Let $m \in \mathbb{N}$ and $t \in T_\Gamma$ such that

$$t' = f(x_0, f(x_1, \dots f(x_{2m}, f(x_{2m+1}, f(x_{2m+2}, \bot, \square_r), \square_r), a_r) \dots), a_r), q_s)$$

matches the term t. Obviously, $\mathcal{A}_{\Sigma_{P'},R_{P'}} \models init_m(t)$.

Let us now suppose that $\mathcal{A}_{\Sigma_{P'},R_{P'}} \models init_m(t)$. There exists $t_1, t_2, s_1, \ldots, s_{2m+1}, s' \in T(\Sigma_P)$ such that

$$
\begin{array}{llll}
t \to t_1 & \wedge\ match[\overline{f}(x_1, \bot, \Box_r)](t_1) & & \\
\wedge\ t_1 \to t_2 & \wedge\ match[A](t_2) & \wedge\ t_2 \to s_1 & \wedge\ match[B](s_1) \\
\wedge\ s_1 \to s_2 & \wedge\ match[C](s_2) & \wedge\ s_2 \to s_3 & \wedge\ match[B](s_3) \\
\ldots & & & \\
\wedge\ s_{2m-1} \to s_{2m} & \wedge\ match[C](s_{2m}) & \wedge\ s_{2m} \to s_{2m+1} & \wedge\ match[B](s_{2m+1}) \\
\wedge\ s_{2m+1} \to s' & \wedge\ match[D](s') & &
\end{array}
$$

Only rules 1-5 are involved in these rewritings. One can easily deduce that the pattern $f(x_0, f(x_1, \ldots f(x_{2m}, f(x_{2m+1}, f(x_{2m+2}, \bot, \Box_r), \Box_r), a_r) \ldots), a_r), q_s)$ matches the term t. $\Box$

Let us now define the formulae *trans* and *final* which checks that a term satisfies Conditions COMPUT. and FINAL of Proposition 16. Afterwards we define for every natural number m the formula $halts_m$ which codes the halting problem of Turing machine U with input w consisting of $2m$-times the symbol a into a formula of the $\exists^*\forall^*$ fragment. Finally we deduce the proof of Theorem 18.

Definition 40.

$$
\begin{array}{rcl}
trans(x) & := & \bigwedge_{p \in P'_T} \neg match[p](x) \\
final(x) & := & match[f(\bot, x_1, q_f)](x) \\
\textit{For all } m \in \mathbb{N}: \ halts_m & := & \exists x \ \big(grid(x) \wedge init_m(x) \wedge trans(x) \wedge final(x)\big)
\end{array}
$$

Let us recall that P'_T denotes the set of linear terms representing the patterns from Figure 4.

Proof of Theorem 18. Let $\Sigma_P \supseteq \Sigma$ and $R_P \supseteq R$ be constructed according to Definitions 20 and 21 where P contains all the patterns used in the above constructions. Let $t \in T_{\Sigma_P}$. We deduce from Lemmas 33–39 that for every natural number m:

$$
\begin{array}{c}
\mathcal{A}_{\Sigma_P,R_P} \models \neg impure(t) \wedge init_m(t) \wedge trans(t) \wedge final(t) \\
\text{if and only if} \\
t \text{ is pure and satisfies conditions of Proposition 16} \\
\text{with input } w \text{ consisting of } 2m\text{-times the symbol } a.
\end{array}
$$

Hence according to Lemma 37 and Proposition 16, for every natural number m:

$$
\begin{array}{c}
\mathcal{A}_{\Sigma_P,R_P} \models grid(t) \wedge init_m(t) \wedge trans(t) \wedge final(t) \\
\text{if and only if} \\
t \text{ is a } \Gamma\text{-grid-term coding a successful computation of Turing machine } U \\
\text{on the input consisting of } 2m\text{-times the symbol } a.
\end{array}
$$

We deduce that given a natural number m, there exists a successful computation of the Turing machine U starting with tape consisting of $2m$-times the symbol a if and only if $\mathcal{A}_{\Sigma_P,R_P} \models halts_m$. This completes the proof of Theorem 18 since the problem considered for u is undecidable.

5 Automata

In this section, we consider the blank-halting problem for the class of Turing machines defined in Section 3.3. First we use the undecidability of this problem to

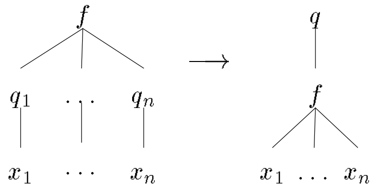
show that the emptiness property for *Trafic*-automata is undecidable (Section 5.2). In fact, it is simple to see that as soon as equality tests between first cousins are available, it is possible to recognize grid-terms (Lemma 45). Local tests (or forbidden patterns) used to describe computations can be realized by usual automata (Lemma 47) because of the linearity of the terms representing patterns. Hence, undecidability simply follows using the good closure properties of automata.
Next we consider the recognizability problem. This problem consists in deciding whether a language of a given class of languages is regular (a language is regular if it is recognizable by a tree automaton without test). In section 5.2, we show that this problem is undecidable for the class of languages recognized by *Trafic*-automata.

5.1 Preliminaries

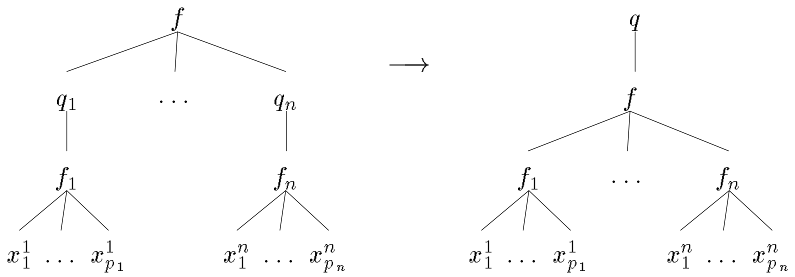
In the sequel we use some properties of regular tree languages. The reader is referred to [9] for proofs of the next lemma and further reading about tree automata.

Lemma 41. *The class of languages recognizable by finite tree automata also called regular tree languages is closed under intersection, union and complement.*
Given a linear pattern π, the tree languages $\{t \mid \pi$ matches $t\}$ and $\{t \mid \pi$ doesn't match $t\}$ are regular.

Definition 42. *A Trafic-automaton is a 4-tuple $\mathcal{A} = (\Gamma', Q_s, F, R)$ where Γ' is a finite ranked alphabet; Q_s is a finite set of unary letters called states which is disjoint from Γ; $F \subseteq Q_s$ is a set of final states; and R is a finite set of transition rules of the following form:*



where $f \in \Gamma'$, $q_1, \dots, q_n, q \in Q_s$ and $x_1, \dots, x_n$ are distinct variables,



where $f, f_1, \dots, f_n \in \Gamma'$, $q_1, \dots, q_n, q \in Q_s$ and $x^k_{i_j} \in \mathcal{X}$.

Note that rules of the second form may be non-linear, that is the same variable may occur twice or more often in a left hand side.
Similarly to the case of tree automata, the tree language recognized by a *Trafic*-automaton is the set of terms of $T_{\Gamma'}$ which are reduced by the rewrite system R into a final state. More formally, we define the relation $\mapsto$ by: $\forall t \in T_{\Gamma' \cup Q_s}$ $t \mapsto t[u]_p$ if and only if there exists a transition rule $l \to r$ in R and substitution σ from $\mathrm{Var}(l)$ into $T_{\Gamma'}$ such that $\sigma(l) = t|_p$ and $\sigma(r) = u$.
Let $\overset{*}{\mapsto}$ be the reflexive and transitive closure of $\mapsto$. The language recognized by the *Trafic*-automaton $\mathcal{A}$ is the set

$$\mathcal{L}(\mathcal{A}) = \{t \in T_{\Gamma'} \mid t \overset{*}{\mapsto} q(t) \text{ where } q \in F\}.$$

Proposition 43. *(i) The class of languages recognized by Trafic-automata is closed under union and intersection.*

(ii) Languages recognized by tree automata are also recognized by Trafic-automata.

Proof. (i) By a construction of a product automaton similar to the case of finite tree automata. The proof is straightforward [25].

(ii) It is obvious that finite tree automata are also *Trafic*-automata.

□

5.2 The Emptiness Problem

Theorem 44. *The emptiness problem for Trafic-automata is undecidable.*

We are going to reduce the halting problem for the class of Turing machines defined in Section 3.3 to the emptiness problem in the class of *Trafic*-automata.
In fact, given a Turing machine $T = (Q, \{a, b\}, \{a, b, \square\}, I, q_s, \square, \{q_f\})$ of this class, we will build a *Trafic*-automaton $\mathcal{A}_{halts}$ such that $\mathcal{L}(\mathcal{A}_{halts})$ is the set of terms coding successful computations of T on the empty input.
In the proof, we first state in Lemma 46 that Γ-grid-terms are recognizable by Trafic-automata. Therefore Lemma 47 proves that codes of a successful computation are also recognizable by Trafic-automata.

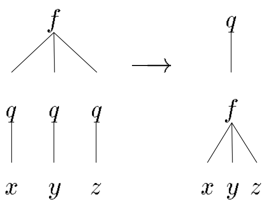
In order to prove the Lemma 46, we first prove the following lemma.

Lemma 45. *There exists a Trafic-automaton $\mathcal{A}_g$ such that*

$$\mathcal{L}(\mathcal{A}_g) = \{t \in T_\Gamma \mid t \text{ satisfies Condition } \textsc{Equality} \text{ of Definition } 2\}.$$

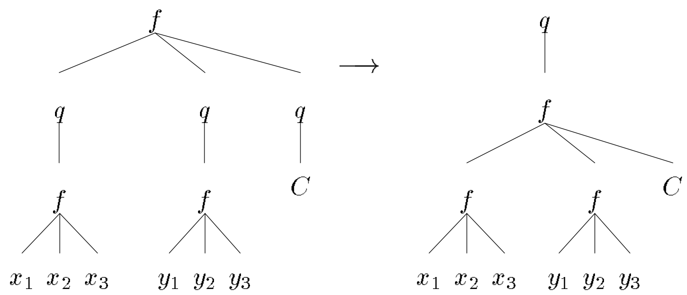
Proof. Let $\mathcal{A} = (\Gamma, \{q\}, \{q\}, R)$ be a *Trafic*-automaton and t be a term of T_Γ of the form $f(f(t_1, t_2, t_3), f(t_4, t_5, t_6), t_7)$ with $t_2 \neq t_4$ and $\forall i \in \{1, \ldots, 7\}, t_i \overset{*}{\mapsto} q(t_i)$. Therefore a rule of $\mathcal{A}$ applies to t if :

- Either the rule is of the form



  where x, y, z are distinct variables,

- Or the rule is of form



where $x_2 \neq y_1$ and C is a constant or of the form $f(z_1, z_2, z_3)$.

with $x_2 \neq y_1$ and C is a constant or of the form $f(z_1, z_2, z_3)$.

Let us now consider $\mathcal{A}_g = (\Gamma, \{q\}, \{q\}, R)$ the *Trafic*-automaton whose set of rules contains all possible rules except the two previous forms of rules. Obviously

$$\mathcal{L}(\mathcal{A}_g) \subseteq \{t \in T_\Gamma \mid t \textit{ satisfies Condition } \textsc{Equality} \textit{ of Definition 2}\}.$$

Moreover one can easily prove that every term of T_Γ of the $f(f(t_1, t_2, t_3), f(t_4, t_5, t_6), t_7)$ with $t_2 = t_4$ is recognized by $\mathcal{A}_g$. Hence

$$\mathcal{L}(\mathcal{A}_g) = \{t \in T_\Gamma \mid t \textit{ satisfies Condition } \textsc{Equality} \textit{ of Definition 2}\}.$$

□

Lemma 46. *There exists a Trafic-automaton $\mathcal{A}_{grid}$ such that*

$$\mathcal{L}(\mathcal{A}_{grid}) = \{t \in T_\Gamma \mid t \text{ is a } \Gamma\text{-grid-term}\}.$$

Proof. All conditions of Definition 2 except Equality correspond to local tests on subterms. Hence, the set L_{gl} of terms satisfying Conditions Edge-1, Edge-2, Constants, Shape-1 and Shape-2 of Definition 2 is (regular and) recognizable by a tree automaton we denote by $\mathcal{A}_{gl}$. 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

$$\mathcal{L}(\mathcal{A}_{grid}) = \mathcal{L}(\mathcal{A}_g) \cap \mathcal{L}(\mathcal{A}_{gl}) = \{t \in T_\Gamma \mid t \text{ is a } \Gamma\text{-grid-term}\}.$$

□

Lemma 47. *There exists a Trafic-automaton $\mathcal{A}_{halts}$ such that*

$$\begin{aligned}\mathcal{L}(\mathcal{A}_{halts}) \quad = \quad & \{t \in T_\Gamma \mid t \textit{ is a } \Gamma\textit{-grid-term coding a successful} \\ & \qquad \textit{computation of the Turing machine } T \textit{ on the empty input}\}.\end{aligned}$$

Proof. Since each term p of P'_T is linear, the set $\{t \in T_\Gamma \mid p \text{ matches } t\}$ is regular. Moreover the languages $\{t \in T_\Gamma \mid f(x_1, f(x_2, f(x_3, \perp, \Box_r), \Box_r), q_s) \text{ matches } t\}$ and $\{t \in T_\Gamma \mid f(\perp, x_1, q_s) \text{ matches } t\}$ are also regular. Therefore according to Lemma 41 there exists tree automata $\mathcal{A}_{init}$, $\mathcal{A}_{trans}$ and $\mathcal{A}_{final}$ such that:

$$\begin{aligned}\mathcal{L}(\mathcal{A}_{init}) &= \{t \in T_\Gamma \mid t \text{ satisfies Conditions } \textsc{Init}(w) \text{ of Proposition 16 with } w = \epsilon\} \\ \mathcal{L}(\mathcal{A}_{trans}) &= \{t \in T_\Gamma \mid t \text{ satisfies Condition } \textsc{Comput}. \text{ of Proposition 16}\} \\ \mathcal{L}(\mathcal{A}_{final}) &= \{t \in T_\Gamma \mid t \text{ satisfies Condition } \textsc{Final} \text{ of Proposition 16}\}\end{aligned}$$

We deduce from Propositions 43 and 16 that there exists a *Trafic*-automaton $\mathcal{A}_{halts}$ such that:

$$\begin{aligned}\mathcal{L}(\mathcal{A}_{halts}) &= \mathcal{L}(\mathcal{A}_{grid}) \cap \mathcal{L}(\mathcal{A}_{init}) \cap \mathcal{L}(\mathcal{A}_{trans}) \cap \mathcal{L}(\mathcal{A}_{final}) \\ &= \{t \in T_\Gamma \mid t \text{ is a } \Gamma\text{-grid-term coding a successful computation} \\ &\qquad \text{of } T \text{ on the empty input}\}\end{aligned}$$

□

The Turing machine T halts on the empty input if and only if $\mathcal{L}(\mathcal{A}_{halts}) \neq \emptyset$ which ends the proof of Theorem 44.

5.3 The Recognizability Problem

Let $\mathcal{CL}$ be a class of languages. The recognizability problem for $\mathcal{CL}$ is the following:

Input: $\mathcal{L} \in \mathcal{CL}$
Question: Is $\mathcal{L}$ regular?

This question is not obvious to decide. In the case of classes of languages recognized by automata with tests it consists in deciding whether the tests are necessary to define a language. In [2], the recognizability problem is shown decidable for the class of languages recognized by automata with tests between brothers [3]. In this section, we use the proof of Theorem 44 to show that the recognizability problem is undecidable as soon as equality tests between cousins are allowed.

Theorem 48. *The recognizability problem is undecidable for the class of languages recognized by* Trafic*-automata.*

Let T be a Turing machine of the class $\mathcal{MT}$ of Turing machines defined in Section 3 and $\mathcal{A}_{halts} = \{\Gamma, Q_s, F, R\}$ be the *Trafic*-automaton recognizing the set of Γ-grid-terms coding successful computations of T on the empty input (Lemma 47).
Let g and h be two new symbols, respectively of arity 1 and 2, and let q_t and q_f two new states. We define a new *Trafic*-automaton $\mathcal{A} = \{\Gamma \cup \{g, h\}, Q_s \cup \{q_t, q_f\}, \{q_f\}, R \cup R'\}$ where R' is:

$$\begin{array}{rcl} g(q(x)) & \rightarrow & q_t(g(x)), \ \ \forall q \in F \\ g(q_t(x)) & \rightarrow & q_t(g(x)) \\ h(q_t(g(x)), q_t(g(x))) & \rightarrow & q_f(h(g(x), g(x))) \end{array}$$

Lemma 49. *The Turing machine T halts on the empty input if and only if $\mathcal{L}(\mathcal{A})$ is not regular.*

Proof. If T halts on the empty input, the language

$$\begin{array}{rcl} \mathcal{L}(\mathcal{A}, q_t) & = & \{t \in T_\Gamma \mid t \overset{*}{\mapsto} q_t(t)\} \\ & = & \{h(g^n(t'), g^n(t')) \mid n \in \mathbb{N}, n \neq 0, t' \in \mathcal{L}(\mathcal{A}_{halts})\} \end{array}$$

is infinite. This language is obviously non regular, therefore $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_f)$ is non regular.
But if T doesn't halt on the empty input, $\mathcal{L}(\mathcal{A}, q_t)$ is empty, then $\mathcal{L}(\mathcal{A})$ is regular. $\square$

Since the blank-halting problem is undecidable for Turing machine, we deduce Theorem 48 from the previous lemma.

6 First-Order Theory of Set Constraints

Let Σ be a signature and $\mathcal{X}$ be a set of variables. A *set expression* is a term of $T_{\Sigma \cup \{\cup, \cap, \complement\}}(\mathcal{X})$ where the arity of symbols $\cup, \cap, \complement$ is respectively 2, 2, 1. A *set constraint* is an expression $\exp_1 \subseteq \exp_2$ where $\exp_1$ and $\exp_2$ are set expressions.
The first-order theory of set constraints is the set of first-order formulae whose atomic formulae are set constraints. The structure $\mathcal{T}_{\mathrm{SC}}(\Sigma)$ associated with this theory is defined as follows: its universe is the set $2^T(\Sigma)$, the symbols $\cup, \cap, \complement$ are interpreted by their usual meaning, and if S and T are two elements of $2^T(\Sigma)$, then $S \subseteq T$ holds in $\mathcal{T}_{\mathrm{SC}}(\Sigma)$ ($\mathcal{T}_{\mathrm{SC}}(\Sigma) \models S \subseteq T$) if and only if S is a subset of T.
The first-order theory of set constraints is undecidable because of the undecidability of the monadic theory of finitely generated free algebra [27]. (One can obviously define a formula which states that a set is a singleton set). The existential fragment is decidable [13, 8, 1] and algorithms based on tree automata have been presented [13]. We pursue these studies and we prove the following theorem:

Theorem 50. *The satisfiability problem of the $\exists^*\forall^*$-fragment of first-order theory of set constraints where only the union ($\cup$) occurs is undecidable.*

We consider the blank-halting problem for the class of Turing machines defined in Section 3.3. Let $T = (Q, \{a, b\}, \{a, b, \Box\}, I, q_s, \Box, \{q_f\})$ be a Turing machine of this class. First we prove Lemma 56 which states that there exists a formula $grid(X)$ in the $\exists^*\forall^*$-fragment of first-order theory of set constraints such that $grid(X)$ is satisfiable if and only if X denotes a singleton set containing a Γ-grid-term. Next following the construction of Section 5, in Definition 59 we define a formula *halts* which is satisfiable if and only if the machine T halts on the empty input.

For the sake of brevity, we define the following formulae:

$$\begin{array}{rcl} empty(X) & := & X \subseteq a_r \wedge X \subseteq b_r \\ disjoint(X,Y) & := & \forall Z\ (Z \subseteq X \wedge Z \subseteq Y) \Rightarrow empty(Z) \\ sing(X) & := & \neg empty(X) \wedge (\forall Z\ Z \subseteq X \Rightarrow empty(Z) \vee X \subseteq Z) \end{array}$$

Obviously, for any sets S and T, $empty(S)$ holds if and only if S is empty, $sing(S)$ holds if and only if S is a singleton and $(disjoint(S,T))$ holds if and only if S and T are disjoint.

Definition 51. *Let C be the set $\Gamma_0 \backslash \{\bot\}$ and grid(X) the formula defined as follows:*

$$grid(X) := sing(X) \wedge \big(\exists S\ subterms(X,S) \wedge equality(S) \wedge shape1(S) \wedge shape2(S)\big)$$

where

$$\begin{array}{rcl} C & := & \bigcup_{c \in \Gamma_0 \backslash \{\bot\}} \{c\} \\ subterms(X,S) & := & S \subseteq f(S,S,C) \cup \bot\ \wedge\ X \subseteq S \\ & & \quad \wedge \forall S'\ (S' \subseteq f(S',S',C) \cup \bot \wedge X \subseteq S') \Rightarrow S \subseteq S' \\ equality(S) & := & \forall Y_1,Y_2,Y_3,Y_4,Y_5,Y_6 (\bigwedge_{i \in \{1,\ldots,7\}} \neg empty(Y_i) \\ & & \quad \wedge f(f(Y_1,Y_2,Y_3), f(Y_4,Y_5,Y_6), Y_7) \subseteq S) \Rightarrow Y_2 = Y_4 \\ shape1(S) & := & \forall Y_1,Y_2,Y_3,Y_4 \wedge disjoint(f(f(Y_1,C,Y_2),Y_3,Y_4),S) \\ shape2(S) & := & \forall Y_1,Y_2,Y_3,Y_4,Y_5,Y_6 \\ & & \quad \wedge disjoint(f(C,f(f(Y_1,Y_2,Y_3),Y_4,Y_5),Y_6),S) \end{array}$$

Let us remark that $grid(X)$, when rewritten in prenex form, is a formula in the $\exists^*\forall^*$-fragment of first-order theory of set constraints. Let us prove now that $grid(X)$ is satisfied if and only if X denotes a singleton set containing a Γ-grid-term (Lemmas 52, 53, 54, 55 and 56). The proofs of Lemmas 52, 53 and 54 are straightforward according to the definitions of formulae.

Lemma 52. *Let $X, S \in 2^{T_\Gamma}$. We $\mathcal{T}_{SC}(\Gamma) \models sing(X)\ \wedge\ subterms(X,S)$ if and only if there exists $t \in T_\Gamma$ satisfying Conditions* Edge-1, Edge-2 *and* Constants *of Definition 2 such that $X = \{t\}$ and S is the set of subterms of t except the constants of $\Gamma_0 \setminus \{\bot\}$.*

Lemma 53. *Let $S \in 2^{T_\Gamma}$. We have $\mathcal{T}_{SC}(\Gamma) \models shape1(S)$ if and only if each term $f(f(t_1,t_2,t_3),t_4,t_5) \in S$ satisfies Condition* Shape-1 *of Definition 2.*

Lemma 54. *Let $S \in 2^{T_\Gamma}$. We have $\mathcal{T}_{SC}(\Gamma) \models shape2(S)$ if and only if each term $f(t_1, f(f(t_2,t_3,t_4),t_5,t_6),t_7) \in S$ satisfies Condition* Shape-2 *of Definition 2.*

Lemma 55. *Let $S \in 2^{T_\Gamma}$. We have $\mathcal{T}_{SC}(\Gamma) \models equality(S)$ if and only if each term $f(f(t_1,t_2,t_3), f(t_4,t_5,t_6), t_7) \in S$ satisfies Condition* Equality *of Definition 2.*

Proof. Let $S \in 2^{T_\Gamma}$ and let us suppose $\mathcal{T}_{\mathrm{SC}}(\Gamma) \models equality(S)$. Therefore $\forall t_1,\ldots,t_7 \in T_\Gamma$:

$$\begin{aligned} f(f(t_1,t_2,t_3), f(t_4,t_5,t_6), t_7) \in S \Rightarrow \\ f(f(\{t_1\},\{t_2\},\{t_3\}), f(\{t_4\},\{t_5\},\{t_6\}), \{t_7\}) \subseteq S \\ &\Rightarrow \{t_2\} = \{t_4\} \\ &\Rightarrow t_2 = t_4 \end{aligned}$$

Let us now suppose that each term $f(f(t_1,t_2,t_3), f(t_4,t_5,t_6), t_7) \in S$ satisfies Condition Equality of Definition 2. Let $Y_1,\ldots,Y_7$ be non empty sets of 2^{T_Γ} such that $f(f(Y_1,Y_2,Y_3), f(Y_4,Y_5,Y_6), Y_7) \subseteq S$.
According to Condition Equality, for any $t_1 \in Y_1,\ldots,t_7 \in Y_7$, we have $t_2 = t_4$ since $f(f(t_1,t_2,t_3), f(t_4,t_5,t_6), t_7) \in S$. We deduce that Y_2 and Y_4 are singletons and $Y_2 = Y_4$. Hence $\mathcal{T}_{\mathrm{SC}}(\Gamma) \models equality(S)$. □

As a corollary to the last four lemmas, we obtain

Lemma 56. *A term $t \in T_\Gamma$ is a Γ-grid-term if and only if $\mathcal{T}_{SC}(\Gamma) \models grid(\{t\})$.*

In the sequel, we will use Proposition 16 and build three formulae denoted by *init*, *trans* and *final* corresponding to conditions of this proposition (Definition 59). To do that, we first need to express by a formula that a pattern π matches a term t. The formulae $match[\pi](S)$ will be satisfiable only when S contains a ground instance of the pattern π. If S is the set of all subterms of t, then the formulae $match[\pi](S)$ will be satisfiable if and only if π matches t.

Definition 57 ($match[\pi](Z)$). *For each linear term $\pi \in \Gamma$ with n variables, we define the following formula:*

$$match[\pi](Z) \ := \ \exists X_1,\ldots,X_n \ \ \pi[X_1,\ldots,X_n] \subseteq Z \wedge \neg empty(\pi[X_1,\ldots,X_n]).$$

Lemma 58. *For each linear term $\pi \in \Gamma$ with n variables and for each $S \in 2^{T_\Gamma}$, $\mathcal{T}_{SC}(\Gamma) \models match[\pi](S)$ if and only if S contains a ground instance of π.*

Proof. Note that it is not necessary to enforce $X_1,\ldots,X_n$ to be singleton sets since we are only interested in satisfiability. The right to left direction is straightforward: if there exists a ground instance $\pi[t_1,\ldots,t_n]$ of π, then take $X_i = \{t_i\}$ and obviously $\mathcal{T}_{\mathrm{SC}}(\Gamma) \models match[\pi](S)$.
To prove the converse, since $\mathcal{T}_{\mathrm{SC}}(\Gamma) \models match[\pi](S)$, take n terms $t_i \in X_i$. Such t_i exists because $\pi[X_1,\ldots,X_n]$ is non empty. Therefore the ground instance $\pi[t_1,...,t_n]$ of π is in S. □

We define now the formulae *init*, *trans* and *final*, and the formula *halts* which codes the halting problem of Turing machines on the empty input into a formula of the $\exists^*\forall^*$ fragment of the first order theory of set constraints.

Definition 59.

$$\begin{aligned} init(Z) &:= match[f(x_1, f(x_2, f(x_3, \perp, \square_r), \square_r), q_s)](Z) \\ trans(Z) &:= \bigwedge_{p \in P'_T} \neg match[p](Z) \\ final(Z) &:= match[f(\perp, x_1, q_f)](Z) \\ halts &:= \exists X\ sing(X) \wedge \big(\exists Z\ subterms(X,Z) \wedge equality(Z) \wedge shape1(Z) \\ &\quad \wedge shape2(Z) \wedge init(Z) \wedge trans(Z) \wedge final(Z)\big) \end{aligned}$$

Proof. Theorem 50. We deduce from Lemma 58 that for each $S \in 2^{T_\Gamma}$:

- $\mathcal{T}_{SC}(\Gamma) \models init(S)$ if and only if the set S contains a ground instance of the term $f(x_1, f(x_2, f(x_3, \bot, \square_r), \square_r), q_s)$ (Conditions INIT(w) of Proposition 16),
- $\mathcal{T}_{SC}(\Gamma) \models trans(S)$ if and only if S contains no ground instance of the patterns of P'_T (Condition COMPUT. of Proposition 16),
- $\mathcal{T}_{SC}(\Gamma) \models final(S)$ if and only if S contains a ground instance of the term $f(\bot, x_1, q_f)$ (Condition FINAL of Proposition 16).

We deduce from these properties, Lemma 56 and Proposition 16 that formula *halts* is satisfiable if and only if there exists a Γ-grid-term satisfying the conditions of Proposition 16 with the empty input. Therefore formula *halts* is satisfiable if and only if there exists a successful computation of Turing machine T starting with the blank tape.
Moreover one of the prenex forms of formula *halts* belongs to the fragment $\exists^*\forall^*$. Therefore we deduce Theorem 50 from the undecidability of the blank-halting problem for Turing machines. □

7 Conclusion

Our reformulation of the grid in the finite tree case allows us to prove three undecidability results in three various domains. We think that this reformulation could be used to prove other undecidability results.

Acknowledgement

We thank Nachum Dershowitz and Joachim Niehren for interesting discussions. We thank the anonymous referees and in particular whom make remarks on Theorem 18 allowing us to improve it.

References

[1] AIKEN, A., KOZEN, D., AND WIMMERS, E. Decidability of systems of set constraints with negative constraints. *Information and Computation 122*, 1 (1995), 30–44.

[2] BOGAERT, B., SEYNHAEVE, F., AND TISON, S. The recognizability problem for tree automata with comparisons between brothers. In *Proceedings of Foundations of Software Science and Computation Structures* (Amsterdam, 1999), vol. 1578 of *Lecture Notes in Computer Science*, pp. 150–164.

[3] BOGAERT, B., AND TISON, S. Equality and disequality constraints on direct subterms in tree automata. In *9th Annual Symposium on Theoretical Aspects of Computer Science* (1992), P. Enjalbert, A. Finkel, and K. W. Wagner, Eds., vol. 577 of *Lecture Notes in Computer Science*, pp. 161–171.

[4] BÖRGER, E., GRÄDEL, E., AND GUREVICH, Y. *The Classical Decision Problem.* Springer Verlag, 1997.

[5] CARON, A., COMON, H., COQUIDÉ, J., DAUCHET, M., AND JACQUEMARD, F. Pumping, cleaning and symbolic constraints solving. In *Proceedings, International Colloquium Automata Languages and Programming* (1994), vol. 820 of *Lecture Notes in Computer Science*, pp. 436–449.

[6] CARON, A.-C., SEYNHAEVE, F., TISON, S., AND TOMMASI, M. Deciding the satisfiability of quantifier free formulae on one-step rewriting. In *Proceedings of the tenth International Conference on Rewriting Techniques and Applications* (Trento, Italy, 1999), P. Narendran and M. Rusinowitch, Eds., no. 1631 in Lecture Notes in Computer Science.

[7] Charatonik, W. An undecidable fragment of the theory of set constraints. *Information Processing Letters 68*, 3 (Nov 1998), 147–151.

[8] Charatonik, W., and Pacholski, L. Set constraints with projections are in NEXPTIME. In *Proceedings of the 35^{th} Symp. Foundations of Computer Science* (1994), pp. 642–653.

[9] Comon, H., Dauchet, M., Gilleron, R., , Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.

[10] Comon, H., Haberstrau, M., and Jouannaud, J.-P. Syntacticness, cycle-syntacticness and shallow theories. *Information and Computation 111*, 1 (May 1994), 154–191.

[11] Courcelle, B. The monadic second-order logic of graphs I. recognizable sets of finite graphs. *Information and Computation 85* (1990), 12–75.

[12] Gilleron, R., Tison, S., and Tommasi, M. Some new decidability results on positive and negative set constraints. In *Proceedings, First International Conference on Constraints in Computational Logics* (1994), J.-P. Jouannaud, Ed., vol. 845 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 336–351.

[13] Gilleron, R., Tison, S., and Tommasi, M. Set constraints and automata. *Information and Computation 149* (1999), 1–41.

[14] Heintze, N. *Set Based Program Analysis.* PhD thesis, Carnegie Mellon University, 1992.

[15] Heintze, N., and Jaffar, J. A Decision Procedure for a Class of Set Constraints. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 42–51.

[16] Heintze, N., and Jaffar, J. A finite presentation theorem for approximating logic programs. In *Proceedings of the 17^{th} ACM Symp. on Principles of Programming Languages* (1990), pp. 197–209. Full version in the IBM tech. rep. RC 16089 (#71415).

[17] Kozen, D. Set constraints and logic programming. *Information and Computation 1*, 142 (1998), 2–25.

[18] Läuchli, H., and Savioz, C. Monadic second order definable relation on the binary tree. *Journal of Symbolic Logic 52* (1987), 219–226.

[19] Lewis, H., and Papadimitriou, C. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

[20] Marcinkowski, J. Undecidability of the first order theory of one-step right ground rewriting. In *Proceedings. Eighth International Conference on Rewriting Techniques and Applications* (Sitges, Spain, 1997), H. Comon, Ed., vol. 1232 of *Lecture Notes in Computer Science*, pp. 241–253.

[21] Mongy, J. *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.

[22] Niehren, J., Pinkal, M., and Ruhrberg, P. On equality up-to constraints over finite trees, context unification and one-step rewriting. In *Procedings of the International Conference on Automated Deduction* (Townsville, Australia, 14-17July 1997), vol. 1249 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 34–48.

[23] Rabin, M. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society 141* (1969), 1–35.

[24] Seese, D. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic 53* (1991), 169–195.

[25] Seynhaeve, F. *Automates et Contraintes.* PhD thesis, Université de Lille 1, 1999.

[26] Seynhaeve, F., Tison, S., and Tommasi, M. An undecidable result for automata with equality tests. reduction to the satisfiability problem for set constraints. Tech. Rep. IT. 295, Laboratoire d'Informatique Fondamentale de Lille, Oct. 1996.

[27] Taitslin, M. Some further examples of undecidable theories. *Algebra and Logic 7* (1968), 127–129. Original paper (russian) in Algebra i Logica 6(3):105-111, 1967.

[28] Talbot, J.-M. *Contraintes Ensemblistes Définies et Co-Définies.* PhD thesis, Université de Lille 1, 1998.

[29] Thomas, W. On logics, tilings, and automata (invited lecture). In *Proceedings, International Colloquium Automata Languages and Programming* (1991), vol. 510 of *Lecture Notes in Computer Science*, pp. 441–??

[30] Treinen, R. The first-order theory of one-step rewriting is undecidable. In *Proceedings. Seventh International Conference on Rewriting Techniques and Applications* (1996), H. Ganzinger, Ed., vol. 1103 of *Lecture Notes in Computer Science*, pp. 276–286.

[31] Treinen, R. The first-order theory of linear one-step rewriting is undecidable. *Theorical Computer Science 208* (1998), 149–177.

[32] Vorobyov, S. The first-order theory of one-step rewriting in linear noetherian systems is undecidable. In *Proceedings. Eighth International Conference on Rewriting Techniques and Applications* (Sitges, Spain, 1997), H. Comon, Ed., vol. 1232 of *Lecture Notes in Computer Science*, pp. 254–268.

[33] Vorobyov, S. The undecidability of the first order theories of one-step rew riting in linear canonical systems. Tech. Rep. MPI-I-98-2-009, Max Planck Institute, 1998.