Set Constraints on Regular Terms

Paweł Rychlikowski and Tomasz Truderung*

Institute of Computer Science, Wrocław University
prych@ii.uni.wroc.pl, tt@ii.uni.wroc.pl

Abstract. Set constraints are a useful formalism for verifying properties of programs. Usually, they are interpreted over the universe of finite terms. However, some logic languages allow infinite regular terms, so it seems natural to consider set constraints over this domain. In the paper we show that the satisfiability problem of set constraints over regular terms is undecidable. We also show that, if each function symbol has the arity at most 1, then this problem is EXPSPACE-complete.

1 Introduction

Set constraints are inclusions between expressions denoting sets of terms. They are a natural formalism for problems that arise in program analysis, including type checking, type inference, and approximating the meaning of programs. They were used in analyzing functional [ALW94], logic [AL94], imperative [HJ94] and concurrent constraint programs [CPM99].

The most popular domain for which set constraints were considered is the Herbrand universe, i.e. the set of all finite terms constructed over a given signature. The satisfiability of such constraints was studied by many authors including N. Heintze and J. Jaffar [HJ90], A. Aiken and E. L. Wimmers [AW92], L. Bachmair, H. Ganzinger, U. Waldmann [BGW93], R. Gilleron, S. Tison and M. Tommasi [GTT93] and W. Charatonik and L. Pacholski [CP94]. Set constraints for other domains were also studied ([MGWK96], [ALW94]).

In this paper we consider a variant of set constraints, namely set constraints over the set of (finite and infinite) regular terms. This domain was first introduced in Prolog II [Col82], and now is used in many modern logic programming languages, such as SWI-Prolog [Wie03] or Eclipse [WNS97]. Classical set constraints over the Herbrand universe can be inadequate in analyzing programs written in these languages.

Infinite terms in the context of set constraints were studied by Charatonik and Podelski [CP98], who proved that, for some restricted class of set constraints, which they call co-definite set constraints, the algorithms working for the Herbrand universe also apply to infinite terms. They proved EXPTIME-completeness of the satisfiability problem of co-definite set constraints over infinite regular terms.

* Partially supported by Polish KBN grant 8T11C 04319.

In general, the satisfiability problem in the Herbrand universe is not equivalent to the satisfiability problem over the set of regular terms. Consider, for example, the signature containing one constant c, and one unary function symbol f. Then the set constraints $X \neq \emptyset, X = f(X)$ have no solution in the Herbrand universe, but they have a solution $X = \{f(f(f(\ldots)))\}$ in the set of infinite terms. Even if we forbid negative constraints, the finite and infinite cases differ. Consider the set constraints (with the same signature) consisting of one equation $\overline{X} = f(X)$. It has a solution in the Herbrand universe, but it is not solvable in the universe of regular terms. The reason is that the regular term $t = f(f(f(\ldots)))$ fulfills the equation $t = f(t)$, so the constraint implies that, in any solution, t belongs to X, if and only if t belongs to $\overline{X}$.

In this paper we show that the satisfiability problem for positive set constraints over regular terms is undecidable. The proof is by reduction of the Post Correspondence Problem. Moreover, we show that, if all function symbols have the arity less or equal to 1, this problem is EXPSPACE-complete. These are rather surprising results, since set constraints over the Herbrand universe are EXPTIME-complete in unary case and NEXPTIME-complete when we allow function symbols of any arity [AKVW93] (they stay in NEXPTIME even if we enrich the language, adding negative constraints and projections [CP94]).

2 Preliminaries

2.1 Signatures and Terms

Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \cdots$ be a signature. A function symbol from Σ_n is told to be of the arity n.

In the paper by a *term* we mean a finite or infinite tree with nodes labeled by elements from Σ. If the label of a node belongs to Σ_n, then this node has exactly n ordered sons. A term t_1 is a *subterm* of t_2, if t_1 is a subtree of t_2. A term t is *regular*, if it has only finitely many different subterms. We denote the set of all (finite and infinite) regular terms over Σ by T^R_Σ. For terms $t_1, \ldots, t_n$, we define the term $f(t_1, \ldots, t_n)$ as a tree t, such that the root of t is labeled by f, and the i-th son of the root is t_i, for $i = 1, \ldots, n$.

In order to describe regular terms we introduce a notion of t-graphs. A *t-graph* is a tuple $\langle \Sigma, V, E \rangle$, where V is a set of vertices, Σ is a signature, and $E : V \to \Sigma \times V^*$, such that if $E(v) = \langle f, v_1 \ldots v_n \rangle$, then f is of the arity n. In such a situation we say that v is labeled by f. If V is finite then we say that the t-graph $\langle \Sigma, V, E \rangle$ is finite. We write $v =_E f(v_1, \ldots, v_n)$ instead of $E(v) = \langle f, v_1 \ldots v_n \rangle$. We say that a vertex v_i is the *i-th son* of v, if and only if $v =_E f(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n)$, for some f.

A regular term can be represented by a t-graph with a selected vertex, as it is shown in Figure 1. This correspondence could be defined formally in the following way: a vertex v in a t-graph $G = \langle \Sigma, V, E \rangle$ *describes* a term t, if there is a function h from the subterms of t to V such that $h(t) = v$, and, for every subterm $t' = f(s_1, \ldots, s_n)$ of t, we have $h(t') =_E f(h(s_1), \ldots, h(s_n))$.

For a t-graph G, by $T(G)$ we denote the set of regular terms represented by the vertices of G. We say that a t-graph is *minimal*, if each of its vertices describes a distinct regular term (in Figure 1, G_2 is minimal, whereas G_1 is not). We denote by M^R_Σ the minimal t-graph (usually infinite) such that $T(M^R_\Sigma) = T^R_\Sigma$. It is easy to see that t-graph exists, and is unique up to isomorphism.

2.2 Set Constraints

Positive set constraints are inclusions[1] of the form $E \subseteq E'$, where the expressions E and E' are given by the grammar

$$E \ ::= \ X \mid E \cap E \mid \overline{E} \mid f(E, \ldots, E) \mid \bot,$$

where X stands for a variable from a given set, and f is a function symbol from a given signature Σ. We will use $\top$ as the abbreviation of $\overline{\bot}$, and $E_1 \cup E_2$, as the abbreviation of $\overline{\overline{E_1} \cap \overline{E_2}}$. We will also identify $\overline{\overline{E}}$ with E.

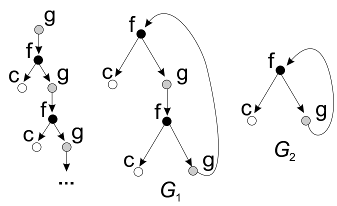

Fig. 1. A regular term (on the left) is represented by gray vertices of G_1 and G_2.

Let SC be a system of set constraints. Let Var denotes the set of variables that appear in SC, and let $\sigma : \mathsf{Var} \to 2^{T^R_\Sigma}$ be an assignment of subsets of T^R_Σ to variables in Var. Then σ in the unique way extends to a function $\hat{\sigma}$ from expressions to subsets of T^R_Σ. This extension is defined as follows: $\hat{\sigma}(X) = \sigma(X)$, for $X \in \mathsf{Var}$, $\hat{\sigma}(\bot) = \emptyset$, $\hat{\sigma}(E_1 \cap E_2) = \hat{\sigma}(E_1) \cap \hat{\sigma}(E_2)$, $\hat{\sigma}(\overline{E}) = T^R_\Sigma \setminus \hat{\sigma}(E)$, and for $f \in \Sigma_n$, we have $\hat{\sigma}(f(E_1, ..., E_n)) = \{f(t_1, \ldots, t_n) \mid t_1 \in \hat{\sigma}(E_1), \ldots, t_n \in \hat{\sigma}(E_n)\}$. An assignment $\sigma : \mathsf{Var} \to 2^{T^R_\Sigma}$ is a solution of SC, if $\hat{\sigma}(E) \subseteq \hat{\sigma}(E')$, for each constraint $E \subseteq E'$ in SC. A system SC of set constraints is satisfiable, if it has a solution.

3 Automata and Set Constraints

We adapt here the definition of a t-dag automaton from [Cha99]. A *t-graph automaton* is a tuple $\langle \Sigma, Q, \Delta \rangle$, where Σ is a finite signature, Q is a finite set of states, and Δ is a set of transitions of the form $f(q_1, \ldots, q_n) \mapsto q$ with $q, q_1, \ldots, q_n \in Q$ and $f \in \Sigma_n$. An automaton is called *complete*, if for each $f \in \Sigma_n$, and each sequence $q_1, \ldots, q_n \in Q$ there exists $q \in Q$ such that $f(q_1, \ldots, q_n) \mapsto q$ belongs to Δ. An automaton $A' = \langle \Sigma, Q', \Delta' \rangle$ is an *subautomaton* of $A = \langle \Sigma, Q, \Delta \rangle$ iff $Q' \subseteq Q$, and $\Delta' \subseteq \Delta$.

A *run* of an automaton $\langle \Sigma, Q, \Delta \rangle$ on a t-graph $G = \langle V, \Sigma, E \rangle$ is a mapping ρ from V to Q such that for each $v, v_1, \ldots, v_n \in V$, and $f \in \Sigma_n$, if $v =_E f(v_1, \ldots, v_n)$, then Δ contains the transition $f(\rho(v_1), \ldots, \rho(v_n)) \mapsto \rho(v)$. If there is a run of an automaton A on a graph G, then we say that A *accepts* G.

The following lemma states the connections between runs on finite graphs and runs on M^R_Σ.

[1] In so called *negative set constraints* there are also allowed negated inclusions. Such systems were analyzed for instance in [CP94].

Lemma 1. *Let Σ be a finite signature, and let A be an automaton over Σ. The following conditions are equivalent:*

(i) *A accepts M^R_Σ.*
(ii) *A accepts all finite t-graphs over Σ.*
(iii) *There exists a complete subautomaton A' of A which accepts M^R_Σ.*
(iv) *There exists a complete subautomaton A' of A which accepts all finite t-graphs over Σ.*

Proof. One can show the following implications: (ii)⇒(i), (i)⇒(iii), (iii)⇒(iv), (iv)⇒(ii). All the implications but the first one are quite straightforward. In the case of $(ii) \Rightarrow (i)$ we can use the compactness theorem for the propositional logic, as is sketched bellow. Assume that (ii) holds. We use the propositional variable p^v_q for each vertex v, and each state q of the given automata A. The intended meaning of p^v_q is "the state q is assigned to the vertex v". For each t-graph G, using these variables, it is easy to construct a set Φ_G of formulas such that Φ_G is satisfiable iff A has a run on G. Now, for any finite $\Phi' \subseteq \Phi_{M^R_\Sigma}$, one can show that there is a finite subgraph G of M^R_Σ such that $\Phi' \subseteq \Phi_G$. By the assumption, A accepts G, so the set Φ_G is satisfiable, and so is Φ'. Hence, by the compactness theorem, $\Phi_{M^R_\Sigma}$ is satisfiable, which implies that A accepts M^R_Σ. □

Following Charatonik and Pacholski [CP94,Cha99] we define, for a system of set constraints SC, the automaton A_{SC} representing it. Let $E(SC)$ be the set of all set expressions occurring in SC together with their complements.

Definition 1. Let SC be a system of set constraints over Σ. The *automaton* A_{SC} is $\langle \Sigma, Q, \Delta \rangle$, where $Q \subseteq 2^{E(SC)}$, and

1. A subset ϕ of E(SC) is a state of A_{SC}, if
   (i) $\perp \notin \phi$,
   (ii) $E \in \phi$ iff $\overline{E} \notin \phi$,
   (iii) if $(E_1 \cap E_2) \in \phi$ then $E_1, E_2 \in \phi$,
   (iv) if $E_1 \in \phi$, $E_2 \in \phi$, and $(E_1 \cap E_2) \in$ E(SC), then $(E_1 \cap E_2) \in \phi$,
   (v) if $E \subseteq E' \in SC$, and $E \in \phi$, then $E' \in \phi$,
   (vi) if $f(E_1, \ldots, E_n) \in \phi$ and $g(E_1, \ldots, E_m) \in \phi$ then $m = n$ and $f = g$,
2. Δ is the set of transitions of the form $f(\phi_1, \ldots, \phi_n) \mapsto \phi$, where
   (i) $f \in \Sigma_n$, and $\phi_1, \ldots, \phi_n, \phi \in Q$, and
   (ii) $f(E_1, \ldots, E_n) \in \phi$ iff $E_i \in \phi_i$ for each $i = 1, \ldots, n$.

The following lemma (and its proof) is an exact 'translation' of the part of Theorem 24 from [Cha99].

Lemma 2. *Let SC be a system of positive set constraints. SC is satisfiable, if and only if A_{SC} accepts M^R_Σ.*

Proof. Suppose that σ is a solution of SC. Let t_v denotes the term described by a vertex v in M^R_Σ. Then we can define a run ρ of A_{SC} on M^R_Σ by setting $\rho(v) = \{E \in E(SC) \mid t_v \in \hat{\sigma}(E)\} \cup \{\overline{E} \mid E \in E(SC), t_v \notin \hat{\sigma}(E)\}$, for each vertex v of M^R_Σ. Conversely, if there exists a run ρ on M^R_Σ, we can define a solution σ of SC by $\sigma(X) = \{t_v \in T^R_\Sigma \mid$ there exists a vertex v of M^R_Σ such that $X \in \rho(v)\}$. □

Notation: As the proof of Lemma 2 shows, solutions and runs express the same relation between terms and variables in different ways: (a) for a solution σ of SC, we write $t_v \in \sigma(X)$, whereas (b) for a run ρ of A_{SC}, we write $X \in \rho(v)$ (where v describes t_v).

We find it convenient to write constraints in a form which directly expresses local relations in a t-graph, using formulas which correspond to (b). So, for $f \in \Sigma_n$, we will write constraints of the form

$$\forall t_0 = f(t_1, \ldots, t_n)) : \quad \varphi, \tag{$\star$}$$

where φ is a boolean combination of formulas of the form $(X \textsf{ in } t_i)$, and $(\overline{X} \textsf{ in } t_i)$ (for $0 \leq i \leq n$).

Constraints of the form ($\star$) can be easily translated to ordinary set constraints: first we change all atomic formulas of the form $(E \textsf{ in } t_0)$ into E, and, for $1 \leq i \leq n$, we change $(E \textsf{ in } t_i)$ into $f(\top, \ldots, E, \ldots, \top)$ (with E on i-th position). Then, we replace $\wedge$ by $\cap$, $\vee$ by $\cup$ and $\neg$ by complementation, obtaining an expression S. The resulting set constraint for ($\star$) is $f(\top, \ldots, \top) \subseteq S$.

For instance the formula $\big(\forall s = f(t_1, t_2) : X \textsf{ in } s \Rightarrow Y \textsf{ in } t_1\big)$ is translated to $f(\top, \top) \subseteq \overline{X} \cup f(Y, \top)$. This formula says that for any run ρ of A_{SC}, for all nodes v and v' such that v is labeled by f and v' is the first son of v, we have that $X \in \rho(v)$ implies $Y \in \rho(v')$.

Moreover, if a sequence of variables $\boldsymbol{X} = (X_1, \ldots, X_n)$ is supposed to code values from some finite set, then we allow to use such vectors of variables as a syntactic sugar in constraints written in the form ($\star$), which is shown in the following example.

Example 1. The formula

$$\forall t = f(s) : (\boldsymbol{X} \textsf{ in } t) \neq (\boldsymbol{X} \textsf{ in } s) \tag{1}$$

is an abbreviation of the formula

$$\begin{aligned}\forall t = f(s) : \quad & (X_1 \textsf{ in } s) \wedge (\overline{X}_1 \textsf{ in } t) \vee (\overline{X}_1 \textsf{ in } s) \wedge (X_1 \textsf{ in } t) \vee \cdots \vee \\ & \qquad (X_n \textsf{ in } s) \wedge (\overline{X}_n \textsf{ in } t) \vee (\overline{X}_n \textsf{ in } s) \wedge (X_n \textsf{ in } t)\end{aligned}$$

Now, the formula (1) means that for any run ρ of the automaton for the constraint (1), for all nodes v and v', such that v' is the only son of v, and v is labeled by f, we have that the value of $\boldsymbol{X}$ in $\rho(v)$ is different than the value of $\boldsymbol{X}$ in $\rho(v')$, i.e. if we take $a_i = 1$ iff $X_i \in \rho(v)$, and $a_i = 0$ iff $\overline{X}_i \in \rho(v)$, and similarly $b_i = 1$ iff $X_i \in \rho(v')$, and $b_i = 0$ iff $\overline{X}_i \in \rho(v')$, then we have that $(a_1 \ldots a_n) \neq (b_1 \ldots b_n)$.

In a similar way we can use formulas like $\forall t = f(s) : (\boldsymbol{X} \textsf{ in } s) = (\boldsymbol{Y} \textsf{ in } t)$ or $\forall t = f(s) : (\boldsymbol{X} \textsf{ in } s = \boldsymbol{X} \textsf{ in } t + 1)$.

4 The General Case

Now we state the main result of the paper. The rest of this section is devoted to its proof.

$$\forall t = a_j(\cdot): \; A \mathit{in}\, t \wedge \overline{S} \mathit{in}\, t \wedge \overline{P} \mathit{in}\, t \tag{2}$$
$$\forall t = s_i(\cdot,\cdot): \; \overline{A} \mathit{in}\, t \wedge S \mathit{in}\, t \wedge \overline{P} \mathit{in}\, t \tag{3}$$
$$\forall t = p_i(\cdot,\cdot,\cdot): \; \overline{A} \mathit{in}\, t \wedge \overline{S} \mathit{in}\, t \wedge P \mathit{in}\, t \tag{4}$$
$$\forall t = a_j(t_1): \; (\mathbf{K} \mathit{in}\, t) = (\mathbf{K} \mathit{in}\, t_1) \tag{5}$$
$$\forall t = p_i(t_1,\cdot,\cdot): \; (\mathbf{K} \mathit{in}\, t) = (\mathbf{K} \mathit{in}\, t_1) \tag{6}$$

$$\forall t = a_j(t_1): \; (E \mathit{in}\, t) \Rightarrow (\overline{S} \mathit{in}\, t_1) \wedge (\overline{A} \mathit{in}\, t_1 \vee E \mathit{in}\, t_1) \tag{7}$$
$$\forall t = p_i(t_1,t_2,t_3): \; (E \mathit{in}\, t) \Rightarrow (\overline{S} \mathit{in}\, t_1) \wedge (E \mathit{in}\, t_1 \vee \overline{P} \mathit{in}\, t_1) \vee (\overline{A} \mathit{in}\, t_2 \vee E \mathit{in}\, t_2) \vee (\overline{A} \mathit{in}\, t_3 \vee E \mathit{in}\, t_3) \vee (\mathbf{K} \mathit{in}\, t) \neq (\mathbf{K} \mathit{in}\, t_2) \vee (\mathbf{K} \mathit{in}\, t) \neq (\mathbf{K} \mathit{in}\, t_3) \tag{8}$$
$$\forall t = s_i(t_1,t_2): \; (E \mathit{in}\, t) \Rightarrow (E \mathit{in}\, t_1) \vee (E \mathit{in}\, t_2) \vee (\overline{P} \mathit{in}\, t_1) \vee (\overline{A} \mathit{in}\, t_2) \tag{9}$$

$$\forall t = s_i(t_1,t_2): \; (H \mathit{in}\, t) \vee (E \mathit{in}\, t) \vee ((\mathbf{K} \mathit{in}\, t) \neq (\mathbf{K} \mathit{in}\, t_1)) \vee ((\mathbf{K} \mathit{in}\, t) \neq (\mathbf{K} \mathit{in}\, t_2)) \tag{10}$$
$$\forall t = a_j(t_1): \; (H \mathit{in}\, t) \Rightarrow (H \mathit{in}\, t_1) \tag{11}$$
$$\forall t = p_i(t_1,t_2,t_3): \; (H \mathit{in}\, t) \Rightarrow (H \mathit{in}\, t_1) \wedge (H \mathit{in}\, t_2) \wedge (H \mathit{in}\, t_3) \tag{12}$$
$$\forall t = s_i(t_1,t_2): \; (H \mathit{in}\, t) \Rightarrow (H \mathit{in}\, t_1 \wedge H \mathit{in}\, t_2) \tag{13}$$

$$\forall t = a_i(t_1): \; (H \mathit{in}\, t) \wedge (\mathbf{U} \mathit{in}\, t) = (a_i x_2 \ldots x_m) \Rightarrow (\mathbf{U} \mathit{in}\, t_1) = (x_2 \ldots x_m) \tag{14}$$
$$\forall t = a_i(\cdot): \; (H \mathit{in}\, t) \wedge (\mathbf{U} \mathit{in}\, t) = \epsilon \Rightarrow (F \mathit{in}\, t) \tag{15}$$

$$\forall t = p_i(t_1,t_2,t_3): \; (H \mathit{in}\, t) \Rightarrow (\overline{C} \mathit{in}\, t) \vee (C \mathit{in}\, t_1) \vee (\mathbf{U} \mathit{in}\, t_2) = u_i \wedge \overline{F} \mathit{in}\, t_1 \vee (\mathbf{U} \mathit{in}\, t_3) = v_i \wedge \overline{F} \mathit{in}\, t_1 \tag{16}$$

$$\forall t = p_i(t_1,t_2,t_3): \; (H \mathit{in}\, t) \wedge \overline{F} \mathit{in}\, t) \Rightarrow (\overline{F} \mathit{in}\, t_2 \wedge \overline{F} \mathit{in}\, t_3 \tag{17}$$

$$\forall t = p_i(t_1,t_2,t_3): \; (H \mathit{in}\, t) \wedge (C \mathit{in}\, t) \Rightarrow (\mathbf{V} \mathit{in}\, t_2) \neq (\mathbf{V} \mathit{in}\, t_3) \tag{18}$$

$$\forall t = s_i(t_1,t_2): \; (H \mathit{in}\, t) \Rightarrow (C \mathit{in}\, t_1) \vee (\mathbf{U} \mathit{in}\, t_2) = u_i \wedge \overline{F} \mathit{in}\, t_1 \vee (\mathbf{U} \mathit{in}\, t_2) = v_i \wedge \overline{F} \mathit{in}\, t_1 \tag{19}$$

Fig. 2. Constraints Φ

Theorem 1. *Deciding whether a system of positive set constraints has a solution in the domain of regular terms is undecidable.*

Let $(u_1, v_1), \ldots, (u_N, v_N)$ be an instance of PCP over an alphabet $\Sigma = \{a_1, \ldots, a_K\}$. We can assume that the words $u_1, v_1, \ldots, u_N, v_N$ are not empty[2]. We give a system Φ of set constraints (Fig. 2) which has a solution, if and only if the given instance of PCP has no solution. The explanation of the the intended meaning of these contstraints will be postponed until Subsections 4.2 and 4.3.

The signature we use consists of the functors s_i of the arity 2, and the functors p_i of the arity 3, for each $1 \leq i \leq N$ (each s_i and p_i corresponds to the pair (u_i, v_i)), and the functors a_j of the arity 1, for each a_j belonging to Σ.

[2] Undecidability of PCP can be proved, if we assume that words are not empty. See e.g. the proof of undecidability of PCP in [HU79].

4.1 H-structures and Solutions of PCP

In this subsection we define a class of finite t-graphs, called *H-structures*, which can be used to code solutions of the given instance of PCP.

We say that a vertex is of the *type* a, if it is labeled by a_j, for some j. Similarly, a vertex is of the *type* s or p, if it is labeled by s_i, p_i respectively, for some i.

Definition 2. An *H-structure* is a t-graph which consists of one vertex x_0 of the type s, called the *root*, nodes $x_1, \ldots x_n$ of the type p, and nodes $y_1, \ldots, y_m$ of the type a such that:
(i) the first son of the root is x_1, and the second son of the root is y_1,
(ii) the only son of y_i is y_{i+1}, for $1 \leq i < m$, and the only son of y_m is x_0,
(iii) for each $1 \leq i < n$, the first son of x_i is x_{i+1}, and the first son of x_n is x_0,
(iv) for each $1 \leq i \leq n$, the second and the third son of x_i are of the type a (thus belong to $\{y_1, \ldots, y_m\}$).

An example of an H-structure is shown in Fig. 3. Let us notice that labels of vertices of the type a correspond to symbols from Σ, thus sequences of vertices of this type can code words over Σ. We formalize it in the following way: a word $w = b_1 \ldots b_n \in \Sigma^*$ *has an instance starting at a vertex* y_1*, and finished at a vertex* y, if there exists a path $y_1, \ldots, y_n$ of vertices labeled by $b_1, \ldots, b_n$, and y is the only son of y_n.

Let x be a vertex of the type p or s. A vertex y is said to be a *second (third) grandson of* x, if y is the second (third) son of the first son of x.

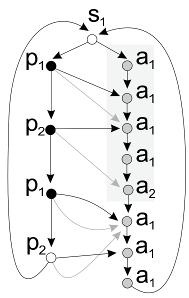

Fig. 3. An H-structure. The first sons of vertices of the type p are represented by down arrows, the second sons by gray right arrow, and the third sons by black right arrow.

Definition 3. A vertex x labeled by p_k is *valid*, if its first son has the type p, and (a) u_k has an instance starting at the second son of x, and finished at its second grandson, and (b) v_k has an instance starting at the third son of x, and finished at its third grandson.

Similarly, a vertex x labeled by s_k is *valid*, if its first son has the type p, and (a) u_k has an instance starting at the second son of x, and finished at its second grandson, and (b) v_k has an instance starting at the second son of x, and finished at its third grandson.

If y is the first son of a valid vertex, then y is told to be *charged*.

Notice, that a charged vertex must have the type p. Now, consider the H-structure from Fig. 3, and suppose that the given instance of PCP has two pairs $(a_1a_1,\, a_1)$, $(a_2,\, a_1a_1a_2)$. Consider the path of vertices labeled by s_1, p_1, p_2, p_1, p_2. The first three vertices of this path are valid. The charged vertices of this H-structure are the black ones.

Definition 4. *An H-structure G with the root x_0 describes a solution*, if there exist charged vertices $x_1, \ldots, x_n$ (of the type p), for $n \geq 1$, such that $x_0, x_1, \ldots, x_n$ is a path in G, and the second and the third son of x_n are the same.

Notice that the vertices $x_0, \ldots, x_{n-1}$ in the definition above must be valid, because the first son of each of them is charged.

Consider again the PCP given by the pairs $(a_1a_1,\ a_1)$, $(a_2,\ a_1a_1a_2)$. The H-structure from Fig. 3 describes a solution of this PCP. The described solution is the sequence $1, 1, 2$. It is given by indices of the labels s_1, p_1, p_2 of the vertices on the path starting with the root. Note that the next vertex on this path with label p_1 (which is not valid, but is charged) is the one whose second and third sons are the same. Note also that the last vertex of the type p is of no use, and we could build a smaller H-structure which describe the same solution.

It is easy to show that the following holds.

Remark 1. The given instance of PCP has a solution, if and only if there exists an H-structure which describes a solution.

Now, we state two lemmas which constitute the major steps of the proof of Theorem 1. The proofs of these lemmas are given in separate subsections.

Lemma 3. *Let Φ be the system of set constraints from Fig. 2. A_Φ accepts all finite t-graphs, if and only if it accepts all H-structures.*

Lemma 4. *Let Φ be the system of set constraints from Fig. 2. A_Φ accepts an H-structure G, if and only if G does not describe a solution.*

Lemmas 2, 3, and 1 imply that system Φ from Fig. 2 is satisfiable, if and only if A_Φ accepts all H-structures. Consequently, by Remark 1 and Lemma 4, the given instance of PCP has no solution, if and only if the system of set constraint from Fig. 2 is satisfiable, which completes the proof of Theorem 1.

4.2 The Proof of Lemma 3

A vertex of the type p is *well-typed*, if its first son has the type p or s, and its second and third sons have the type a. A vertex of the type a is *well-typed*, if its only son has the type a or s. A vertex of the type s is *well-typed*, if its first son has the type p, and its second son has the type a. Notice that all the vertices of any H-structure are well-typed.

Now, we introduce a notion of *witness* which is intended to give an evidence that a vertex v (of the type s) is *not* a root of any H-structure, and carry some information which will be used later in the proof.

Definition 5. Let G be a t-graph, and v be a vertex of the type s. A *witness* for v has one of the following forms:

(a) $\langle v, \{v, z_1, \ldots, z_n\}\rangle$, if $v, z_1, \ldots, z_n$ is a path in G, where none of $z_1, \ldots, z_n$ has the type s, and only the last vertex of $v, z_1, \ldots, z_n$ is not well-typed.

(b) $\langle v, \{v, z_1, z_2, \dots\}\rangle$, if $v, z_1, z_2, \dots$ is an infinite path in G of well-type vertices, where none of $z_1, z_2, \dots$ has the type s.
(c) $\langle w, \{v, x_1, \dots, x_n\}\rangle$, if $v, x_1, \dots, x_n, y_1, \dots, y_m, w$ is a path in G, where the vertex $w \neq v$ has the type s; the vertices $x_1, \dots, x_n$ are well-typed, and have the type p; the vertices $y_1, \dots, y_m$ are well-typed, and have the type a.
(d) $\langle w, \emptyset\rangle$, if there is a path of the form $v, x_1, \dots, x_n, w$ in G, where the vertex $w \neq v$ has the type s; the vertices $x_1, \dots, x_n$ are well-typed, and either each of them has the type p, or each of them has the type a.

The set of all the witnesses for v is denoted by $W(v)$.

Note that a witness of the form (a) corresponds to the case, when starting with v, we can reach some vertex which is not well-typed. A witness of the form (b) corresponds to the case, when there is an infinite path of well-typed vertices of the type p or a starting with v. A witness of the form (c) or (d) corresponds to the case when starting with v, we can reach a vertex $w \neq v$ of the type s: a witness has the form (c) if the path from v to w contains vertices of the type p followed by vertices of the type a, and a witness has the form (d) if the path from v to w contains vertices only of the type p, or only of the type a.

One can check that a vertex v of the type s is the root of some H-structure, if and only if $W(v)$ is empty.

In order to prove the nontrivial implication of Lemma 3, let us assume that A_Φ accepts all H-structures. Let G be a finite t-graph, and G_0 be the subgraph of G containing exactly all the H-structures of G. Because these H-structures have disjoint sets of vertices, and each of them has an accepting run, there exists an accepting run ρ_0 of A_Φ on G_0. We will extend ρ_0 to a run ρ on G, but first we informally explain the role of some variables used in the constraints from Fig. 2:

- A, S, P – type variables. In each vertex v exactly one of these variables have to be set: A has to be set in v (i.e. $A \in \rho(v)$), if v has the type a, and so on (see (2)–(4)).
- $\boldsymbol{K}$ — vectors of variables which can code a *color*, i.e. a value from $\{\alpha, \beta, \gamma\}$. The constraints (5)–(6) guarantee that each vertex of the type a or p has the same color as its first son, thus each H-structure is colored with one color. These variables are used to detect cases related to the points (c) and (d) of Definition 5.
- E – an error flag. If set in a vertex v, it indicates that v cannot be a part of any H-structure. The constraints (7)–(9) allow us to set E in v only if either (i) E is set in some son of v of a type different than s, (ii) v is not well-typed, or (iii) v has the type p and it has a different color than its second or third son (it is related to point (c) of Definition 5).
- H – an H-structure indicator. This variable is intended to be set exactly in these vertices which are a part of some H-structure. The constraints (11)–(13) guarantees that, if H is set in some vertex, then it must be also set in all its sons. The constraint (10) guarantees that H must be set in a vertex v of the type s unless (i) E is set in v, which corresponds to the points (a),

(b) or (c) of Definition 5, or (ii) v has a different color than one of its sons, which corresponds to point the (d) of Definition 5.
Note that, if the variable H is not set in some vertex v (so v is not a part of any H-structure), then (11)–(19) are obviously satisfied in v.

As we have noticed, v of the type s is the root of some H-structure, if and only if $W(v) = \emptyset$. Let V_S denote the set of vertices of the type s from $G \setminus G_0$. For each $v \in V_S$, $W(v) \neq \emptyset$, so let us chose one witness from $W(v)$, and denote it by $f(v)$. One can assign a color $c_v \in \{\alpha, \beta, \gamma\}$ to each vertex $v \in G$ of the type s in such a way that, if $f(v) = \langle w, B \rangle$, for some $w \neq v$, then $c_v \neq c_w$, and, for $v \in G_0$, we have $c_v = (\boldsymbol{K} \textit{ in } \rho_0(v))$.

Now, we extend ρ_0 to a run ρ on G. For each $v \in G \setminus G_0$, we define $\rho(v)$ as follows: We set variable H to 0. We set type variables (A, S, P) according to the type of v (e.g. $A, \overline{S}, \overline{P}$ in vertices of the type a). We set variable E to 1, if and only if, for some $v' \in V_S$ with $f(v') = \langle w, B \rangle$, the vertex $v \in B$. If $v \in V_S$ then we set $\boldsymbol{K}$ to c_v, otherwise we give v the color of its first son (when following first sons, we get into a cycle, then we can set $\boldsymbol{K}$ to α in all the vertices of this cycle). The values of the other variables do not matter. One can now show that ρ is a run on G. □

4.3 The Proof of Lemma 4

Lemma 5. *Let G be an H-structure, and ρ be a run of A_Φ on G. For each vertex v of G, the variable H must be set in $\rho(v)$.*

Proof. It is easy to check that all the vertices of G have the same values of $\boldsymbol{K}$ in ρ. Moreover, one can check that E must not be set in $\rho(v)$, for each v in G. Thus, the only way to satisfy the constraint (10) for the root r is to set H in $\rho(r)$. So, by the constraint (11)–(13), H must be set in all the vertices of G.

Let us now describe the intended meaning of the variables used in this part of the proof. As we consider here H-structures, we assume that, according to Lemma 5, the variable H must be set in any vertex considered.

- $\boldsymbol{U}$ — a sequence of variables of the length sufficient to code a special value $\diamond$, and words over Σ not longer than l, where l is the length of the longest of the words in the given instance of PCP. It is used to check whether some word has an instance at a given place (see (14)–(15)), and so to check validity of vertices.
- F — an auxiliary variable used together with $\boldsymbol{U}$ to check validity of vertices.
- C — the constraints (14)–(16) and (19) guarantee that, in any run, C have to be set in the vertices denoted by $x_1, \ldots, x_n$ in Definition 4 (that is in the sequence of charged vertices).
- $\boldsymbol{V}$ — a vector of variables which can code one of the colors α, β, γ. It is used only in (18) which guarantees that the second and the third son of a charged vertex (a vertex with the variable C set) must not be the same,

In order to prove Lemma 4, we first show that **if an H-structure G describes a solution, then A_Φ does not accept G.**

Proof. Suppose that G describes a solution, and suppose that ρ is a run of A_Φ on G. Let $x_0, \ldots, x_n$ denote vertices according to Definition 4. Using Lemma 5, one can show that (14)–(15) have the following consequence: if a word u has an instance starting at x and finished at y, and $(\boldsymbol{U}\, in\, \rho(x)) = u$, then $F \in \rho(y)$.

Using that together with (16), (17), (19) one can prove, by induction on i, that $C \in \rho(x_i)$, for $1 \leq i \leq n$. Particularly, $C \in \rho(x_n)$. By Definition 4, the second and the third sons of x_n are the same, which implies that ρ cannot fulfill (18), and contradicts the assumption that ρ is a run on G. □

Now, we show that **if an H-structure G does not describe a solution, then A_Φ accepts G.**

Proof. Let $x_0, \ldots, x_n$ and $y_1, \ldots, y_m$ denote vertices according to Definition 2. For $i \in \{1, \ldots, n\}$, we define $s(i)$ and $t(i)$ in such a way that $y_{s(i)}$ is the second son of x_i, and $y_{t(i)}$ is its third son.

Let x_k be the first not valid vertex from $x_0, \ldots, x_n$ (such a vertex exists, because x_n is not valid). Notice that G does not describe a solution, and, for each $i \in \{1, \ldots, k\}$, x_i is charched, so we have $s(i) \neq t(i)$, and moreover, $s(i) < s(i+1)$, and $t(i) < t(i+1)$. Using these facts, one can prove that each vertex v can be assigned a color $f(y) \in \{\alpha, \beta\}$ such that, for each $i \in \{1, \ldots, k\}$, it holds $f(y_{s(i)}) \neq f(y_{t(i)})$.

Now, we will define a function δ from the set of vertices of G to the set of values that can be coded in $\boldsymbol{U}$. If $k = n$, then let $\delta(v) = \diamond$, for each vertex v of G. Otherwise (i.e. if $k < n$), since x_k is not valid, there are two possible cases which correspond to violation of the condition (a), or the condition (b) of Definition 3. We consider the case (a), and we assume that $k \neq 0$ (in the other cases the proof proceeds similarly). Let p_j be the label of x_k, and $u_j = b_0 \ldots b_l$. Let d be the greatest natural number such that the labels of $y_{s(k)}, \ldots, y_{s(k)+d}$ are some prefix of u_j (d must not be greater than l). For $0 \leq i \leq d+1$, let $\delta(y_{s(k)+i}) = b_i \ldots b_l$. For a vertex $v \notin \{y_{s(k)}, \ldots, y_{s(k)+d+1}\}$, let $\delta(v) = \diamond$.

Now we construct a run ρ such that:

- in each vertex v of G we set H to 1 and E to 0; the type variables we set according to the type of v, and the variables $\boldsymbol{K}$ we set to α,
- we set F to 1 in each vertex, with the exception of x_{k+1} and its second and third sons, if $k < n$,
- we set the variable C to 1 only in the vertices $x_0, \ldots, x_k$,
- in each vertex v, we set $\boldsymbol{V}$ to $f(v)$, and $\boldsymbol{U}$ to $\delta(v)$.

One can check that ρ is a run of A_Φ on G. □

5 The Unary Case

In this section we consider positive set constraints which use only constants and unary function symbols. Such systems will be called *unary set constraints (USC)*. The problem of deciding whether a system of USC is satisfiable turns out to be EXPSPACE-complete, which is an immediate consequence of the following theorems.

Theorem 2. *Deciding whether a system of USC is satisfiable is in EXPSPACE.*

Theorem 3. *Deciding whether a system of USC is satisfiable is EXPSPACE-hard.*

5.1 Proof of Theorem 2

Let Σ be a signature, and $f_1, \ldots, f_n \in \Sigma$. A finite t-graph $\langle V, \Sigma, E \rangle$ with the set of vertices $V = \{v_1, \ldots, v_n\}$ is *a cycle* $f_1, \ldots, f_n$ *(over* Σ*)*, if $v_1 =_E f_1(v_n)$, and $v_i =_E f_i(v_{i-1})$, for $1 < i \leq n$.

Intuitively, cycles are the most difficult parts of a t-graph when we want to find a run. This is stated by the following lemma:

Lemma 6. *Let A be a complete automaton over Σ. A accepts all finite t-graphs over Σ, if and only if it accepts all cycles over Σ.*

Proof. Sketch. To proceed the proof in the nontrivial direction note that the connected components of a graph can be considered separately. Each such component contains at most one cycle. To find a run for a whole connected component, we first find a run on the only cycle, and then use the completeness of the automaton.

Now, for a t-graph automaton A, we define a deterministic finite automaton $\tilde{A}$ (working on finite words) in such a way that runs of A on cycles can be simulated by $\tilde{A}$.

Definition 6. *Let $A = \langle \Sigma, Q, \Delta \rangle$ be a t-graph automaton with $Q = \{q_1, \ldots, q_n\}$. We define a deterministic finite automaton $\tilde{A}$ on finite words over the alphabet Σ_1 (i.e. over the set of the unary symbols of Σ) in the following way: let $\tilde{A} = \langle \Sigma_1, \tilde{Q}, \tilde{q}_0, \tilde{\delta}, \tilde{F} \rangle$, where $\tilde{Q} = (2^Q)^n$ is the set of states, $\tilde{q}_0 = \langle \{q_1\}, \ldots, \{q_n\} \rangle$ is the initial state, and $\tilde{F} = \{\langle Q_1, \ldots, Q_n \rangle \mid q_i \in Q_i$ for some $1 \leq i \leq n\}$ is the set of accepting states. The transition function $\tilde{\delta}$ is defined by the equation*

$$\tilde{\delta}(\langle Q_1, \ldots, Q_n \rangle, f) = \langle Q'_1, \ldots, Q'_n \rangle,$$

where $Q'_i = \{q' \in Q \mid$ there exists $q \in Q_i$ such that $(f(q) \mapsto q') \in \Delta\}$.

One can prove the following lemma which expresses a correspondence between automata on words and t-graph automata on cycles.

Lemma 7. *Let A be a t-graph automaton over Σ, and $f_1, \ldots, f_k \in \Sigma$. The automaton A accepts the cycle $f_1, \ldots, f_k$, iff $\tilde{A}$ accepts the word $f_1 \ldots f_k$.*

Deciding whether a deterministic finite automaton is universal (i.e. accepts all words) is NLOGSPACE complete. Knowing that, since the size of $\tilde{A}$ is $2^{O(n^2)}$, it is easy to prove that, for a t-graph automaton A, the problem of deciding whether $\tilde{A}$ is universal is in PSPACE.

Now we give the nondeterministic algorithm working in EXPSPACE, and verifying whether a given system SC of set constraints is satisfiable: *First we* *construct A_{SC}. Then we guess*[3] *a complete subautomaton B of A_{SC} and verify whether $\tilde{B}$ is universal. If it is, we halt with the answer "yes", otherwise halt with the answer "no".* It is easy to check that this algorithm works in EXPSPACE. Its correctness follows directly from Lemmas 1, 2, 6, 7.

5.2 Proof of Theorem 3

Suppose that M is a deterministic Turing machine which, for an input word w of length n ($w \in \{0,1\}^*$), uses space bounded by 2^N where $N = n^l$, for some integer l. We assume that the set of states $Q = \{0, \ldots, K\}$, the tape alphabet $\Gamma = \{0, \ldots, L\}$, the number 0 denotes the initial state and the blank symbol, $Q_F \subseteq Q$ is the set of accepting states, and δ is the transition function.

Without loss of generality we can assume that Q is a union of three disjoint sets: Q_L, Q_R and $\{0\}$, such that M can be in a state belonging to Q_L (Q_R) only after its head has been moved left (right respectively).[4] Thus the transition function δ can be seen as a function from $Q \times \Gamma$ to $Q \times \Gamma$.

Let $u = u_1 \ldots u_n \in \{0,1\}^*$ be the input word. We will construct a system Ψ of set constraints such that M accepts u, if and only if Ψ is not satisfiable. In set constraints Ψ we use the signature $\Sigma = \Sigma_1 = \{f_0, \ldots, f_K\}$.

Let us notice that a sequence $0, q_1, \ldots, q_n$ of states of M can be coded as the cycle $f_0, f_{q_1}, \ldots f_{q_n}$. This gives us opportunity to code computations using t-graphs (note that a sequence of states determines the position of the head). A sequence $0, q_1, \ldots, q_n$ of states of M is *accepting*, if $q_n \in Q_F$. It is *valid*, if there is a computation of M on u with these states.

We will consider a computation of M for u from the point of view of the i-th cell. A sequence $0, q_1, \ldots, q_n$ of states of M is *valid with respect to the i-th cell*, if there exists a sequence of tape symbols $a_0, a_1, \ldots, a_n$, such that (1) a_0 is the tape symbol contained in the i-th cell at the start of computation, and (2) if the position of the head of M in the j-th step is i, then $\delta_M(q_j, a_j) = \langle q_{j+1}, a_{j+1} \rangle$, otherwise $a_{j+1} = a_j$.

Note that a sequence of states is valid, if and only if it is valid with respect to the i-th cell, for all $0 \leq i < 2^N$. In Figure 4 we give a system Ψ of set constraints which is solvable, if and only if M does not accept u.

In the constraints we use the following variables: $\boldsymbol{P} = P_1, \ldots, P_N$ (related to the position of the head), $\boldsymbol{A} = A_1, \ldots, A_N$ (related to the address of a cell), $\boldsymbol{X} = X_1, \ldots, X_{\lceil log_2 L \rceil}$ (related to the content of the cell pointed by A), $\boldsymbol{Q} = Q_1, \ldots, Q_{\lceil log_2 K \rceil}$ (related to a state of M). We use a special variable T which is a sign of invalid computation and variables $\boldsymbol{K}$ which are supposed to code a color (i.e. an element from $\{\alpha, \beta, \gamma\}$).

One can prove the following lemma.

Lemma 8. *A_Ψ accepts all finite t-graphs, if and only if A_Ψ accepts all cycles of the form $f_0, f_{q_1}, \ldots f_{q_n}$ where $q_i \neq 0$, for $i = 1, \ldots, n$.*

[3] We can use nondeterminism here because EXPSPACE=NEXPSPACE.

[4] We can easily transform any Turing machine, so as it meets this condition.

$$\forall t = f_0(s): \quad (\mathbf{K}\,\mathsf{in}\,s) \neq (\mathbf{K}\,\mathsf{in}\,t) \vee \mathsf{start}(t) \wedge \overline{T}\,\mathsf{in}\,t \wedge \;\; T\,\mathsf{in}\,s \vee \mathsf{nacc}(s) \;\; , \tag{20}$$

$$\forall t = f_i(s): \quad (\mathbf{K}\,\mathsf{in}\,t) = (\mathbf{K}\,\mathsf{in}\,s) \wedge \tag{21}$$

$$((T\,\mathsf{in}\,s) \vee (\overline{T}\,\mathsf{in}\,t \wedge \mathsf{good}_i(t,s)) \qquad \text{(for all } 0 < i \leq K), \tag{22}$$
$$\vee\; (T\,\mathsf{in}\,t \wedge \mathsf{bad}_i(t,s)))$$

where

$$\mathsf{nacc}(t) \equiv \bigwedge_{i \in Q_F} (\mathbf{Q}\,\mathsf{in}\,t) \neq i, \tag{23}$$

$$\mathsf{start}(t) \equiv (\mathbf{P}\,\mathsf{in}\,t) = 0 \;\wedge\; (\mathbf{Q}\,\mathsf{in}\,t) = 0 \wedge \tag{24}$$

$$(\mathbf{A}\,\mathit{in}\,t) > n \wedge (\mathbf{X}\,\mathit{in}\,t) = 0 \vee \bigvee_{1 \leq i \leq n} (\mathbf{A}\,\mathit{in}\,t) = i \wedge (\mathbf{X}\,\mathit{in}\,t) = u_i \tag{25}$$

$$\mathsf{bad}_i(t,s) \equiv (\mathbf{A}\,\mathit{in}\,s) = (\mathbf{P}\,\mathit{in}\,s) \;\wedge\; \delta_M((\mathbf{Q}\,\mathit{in}\,s),(\mathbf{X}\,\mathit{in}\,s)) = \langle j, v \rangle, \text{where } j \neq i \tag{26}$$

$$\mathsf{good}_i(t,s) \equiv \mathsf{next}_i(t,s) \;\wedge\; (\mathbf{A}\,\mathit{in}\,t) = (\mathbf{A}\,\mathit{in}\,s) \;\wedge\; (\mathbf{Q}\,\mathit{in}\,t\, = i) \;\wedge\; (\Phi_i \vee \Phi_i') \tag{27}$$

$$\Phi_i \equiv (\mathbf{A}\,\mathit{in}\,s) = (\mathbf{P}\,\mathit{in}\,s) \;\wedge\; \delta_M(\mathbf{Q}\,\mathit{in}\,s, \mathbf{X}\,\mathit{in}\,s) = \langle i, \mathbf{X}\,\mathit{in}\,t \rangle \tag{28}$$

$$\Phi_i' \equiv (\mathbf{A}\,\mathit{in}\,s) \neq (\mathbf{P}\,\mathit{in}\,s) \;\wedge\; (\mathbf{X}\,\mathit{in}\,t) = (\mathbf{X}\,\mathit{in}\,s) \tag{29}$$

$$\mathsf{next}_i(t,s) \equiv \begin{cases} (\mathbf{P}\,\mathit{in}\,t) = (\mathbf{P}\,\mathit{in}\,s) + 1, & \text{if } i \in Q_R \\ (\mathbf{P}\,\mathit{in}\,t) = (\mathbf{P}\,\mathit{in}\,s) - 1, & \text{if } i \in Q_L \end{cases} \tag{30}$$

Fig. 4. Constraints Ψ

Now we give some intuition how A_Ψ works on t-graphs which possibly code computations of M. The main idea is as follows: successive vertices of a t-graph describe successive states of computation of M from the point of view of the k-th cell. The number k is coded in $\boldsymbol{A}$, and $\boldsymbol{X}$ represents the content of the cell within the computation. $\mathsf{next}_i(t,s)$ describes changes of the position of the head, $\mathsf{nacc}(t)$ says that the state coded in vertex t is not accepting. The expression $\mathsf{good}_i(t,s)$ guarantees that consecutive vertices have proper values of $\boldsymbol{P}$, $\boldsymbol{A}$, $\boldsymbol{Q}$, $\boldsymbol{X}$: the value of $\boldsymbol{P}$ is incremented or decremented dependently of the state, the value of $\boldsymbol{A}$ is copied (since we still look at the same cell), and the value of $\boldsymbol{Q}$, which codes a state, changes according to the label of the node. The value of $\boldsymbol{X}$ changes only if the head of the machine M looks at the selected cell. These changes have to agree with the transition function of M. The expression $\mathsf{bad}_i(t,s)$ says that i cannot be the next state of M, if the current state was coded in s.

The next lemma relates cycles of the form $f_0, f_{q_1}, \dots f_{q_n}$ with computations of M.

Lemma 9. *A_Ψ does not accept a cycle $f_0, f_{q_1}, \dots f_{q_n}$ where $q_i \neq 0$ for $i = 1, \dots, n$, if and only the sequence $0, q_1, \dots, q_n$ is valid and accepting.*

Lemmas 8, and 9 suffice to conclude that A_Ψ accepts all finite t-graphs iff M rejects u. So, thanks to Lemmas 1 and 2, M rejects u iff the system Ψ has a solution, which completes the proof of Theorem 3.

6 Future Works

It could be useful to find some other variants of set constraints for which the satisfiability problem over sets of regular terms is decidable. Particularly, it is worth to consider so called definite set constraints [HJ90] for which the satisfiability problem over the Herbrand universe is EXPTIME-complete [CP97].

References

[AKVW93] A. Aiken, D. Kozen, M. Vardi, and E. L. Wimmers, *The complexity of set constraints*, Proceedings of CSL 1993, Springer Verlag, 1993, pp. 1–17.

[AL94] A. Aiken and T.K. Lakshman, *Directional type checking of logic programs*, Proceedings of the First International Static Analysis Symposium (Baudoin Le Charlier, ed.), Springer Verlag, 1994, pp. 43–60.

[ALW94] A. Aiken, T.K. Lakshman, and E. Wimmers, *Soft typing with conditional types*, Proceedings of POPL 1994, ACM Press, 1994, pp. 163–173.

[AW92] A. Aiken and E. Wimmers, *Solving systems of set constraints*, Proceedings of LICS 1992, IEEE Computer Society Press, 1992, pp. 329–340.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann, *Set constraints are the monadic class*, Proceedings of LICS 1993, IEEE Computer Society Press, 1993, pp. 75–83.

[Cha99] W. Charatonik, *Automata on dag representations of finite trees*, Tech. report, Max-Planck-Institut für Informatik, 1999.

[Col82] A. Colmerauer, *Prolog II reference manual and theoretical model*, Universite de la Mediterranee Aix-Marseille II, 1982.

[CP94] W. Charatonik and L. Pacholski, *Set constraints with projections are in NEXPTIME*, FOCS 1994, IEEE Comp. Society Press, 1994, pp. 642–653.

[CP97] W. Charatonik and A. Podelski, *Set constraints with intersection*, Proceedings of LICS 1997, IEEE Computer Society Press, 1997, pp. 362–372.

[CP98] W. Charatonik and A. Podelski, *Co-definite set constraints*, Proceedings of RTA 98, Springer Verlag, 1998, pp. 211–225.

[CPM99] W. Charatonik, A. Podelski, and M. Müller, *Set-based failure analysis for logic programs and concurrent constraint programs*, Proceedings of the European Symposium on Programming, Springer Verlag, 1999, pp. 177–192.

[GTT93] R. Gilleron, S. Tison, and M. Tommasi, *Solving systems of set constraints with negated subset relationships*, Proceedings of FOCS 1993, IEEE Computer Society Press, 1993, pp. 372–380.

[HJ90] N. Heintze and J. Jaffar, *A decision procedure for a class of set constraints*, Proceedings of LICS 1990, IEEE Computer Society Press, 1990, pp. 42–51.

[HJ94] N. Heintze and J. Jaffar, *Set constraints and set-based analysis*, Proceedings of CP 1994, Springer Verlag, 1994, pp. 281–298.

[HU79] J. Hopcroft and J. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.

[MGWK96] D. A. McAllester, R. Givan, C. Witty, and D. Kozen, *Tarskian set constraints*, LICS 1996, IEEE Computer Society Press, 1996, pp. 138–147.

[Wie03] J. Wielemaker, *SWI-Prolog 5.1 reference manual*, 2003.

[WNS97] M. Wallace, S. Novello, and J. Schimpf, *ECLiPSe: A Platform for Constraint Logic Programming*, Tech. report, IC-Parc, Imperial College, London, 1997.