Undecidability of ground reducibility for word rewriting systems with variables

Gregory KUCHEROV and Michaël RUSINOWITCH *

Key words: Theory of Computation; Formal Languages; Term Rewriting Systems; Pattern Matching

1 Introduction

Given a finite alphabet A and an alphabet of variables $\mathcal{X}$, we consider words over $A \cup \mathcal{X}$ called *words with variables* (or *patterns*) and words over A called simply *words* (or *strings*). A finite set $\mathcal{R}$ of rewrite rules $w \to v$ where w, v are words with variables is called a *word rewriting system with variables (WRSV)*. From the viewpoint of rewriting systems theory WRSV's are rewriting systems over a signature consisting of one binary associative function (concatenation) and finitely many constant symbols (letters). If the rules of $\mathcal{R}$ do not contain variables, we obtain *word rewriting systems* (string rewriting systems, semi-Thue systems) that have been investigated for a long time [1].

In this work we are interested in the *reductive power* of WRSV's. For this reason we simply identify a WRSV with the set of its left-hand sides. Also, we sometimes call patterns of $\mathcal{R}$ *rules* without adding the prefix "the left-hand side of".

Let $Inst(\mathcal{R})$ denote the set of instances of patterns from $\mathcal{R}$, that is the set of $\sigma(p)$ for all $p \in \mathcal{R}$ and all substitutions σ of variables by non-empty words[1] (different occurrences of the same variable are substituted by the same word). A pattern p *applies to* (or *reduces*) a word v, if $v \in A^* Inst(\{p\}) A^*$. We are interested in the properties of the set $Red(\mathcal{R}) = A^* Inst(\mathcal{R}) A^*$, that is the set of all words reducible by $\mathcal{R}$. In this paper we study the following *ground reducibility problem*: for a given word with variables w, does $Inst(\{w\}) \subseteq Red(\mathcal{R})$ hold, i.e. are all the instances of w reducible by $\mathcal{R}$? If the inclusion holds, w is said to be *ground reducible* by $\mathcal{R}$.

The ground reducibility problem has been proved decidable for ordinary term rewriting systems [7, 4]. However, in the presence of associative functions the problem is more complex. It was shown undecidable in [3] for term rewriting systems over a signature containing among other functions an associative function. This result, however, cannot be generalized to WRSV's, as the auxiliary non-associative functions are essential in the proof.

The ground reducibility problem can be also stated in the setting of *pattern languages* [8] or (non-linear) pattern matching. (Here "pattern matching" should be understood in the extended sense: a pattern w matches a word v iff v has a factor which is

*CRIN and INRIA-Lorraine, Campus Scientifique, BP 239, F-54506 Vandœuvre-lès-Nancy, France, email: {kucherov,rusi}@loria.fr

[1] The main result of this paper remains valid if we allow variables to be substituted by the empty word.

an instance of w). Thus, the ground reducibility problem could be rephrased as follows: given a set of patterns $\mathcal{R}$, is every string matched by a given pattern w also matched by some pattern from $\mathcal{R}$?

In this note we prove this question to be generally undecidable. It remains undecidable even for a fixed and very simple word $w_0 = axa$ where a is a letter and x a variable.

2 Main Result

We will use the following formalism of *Minsky machines*. A machine $\mathcal{M}$ is operated by a *program* $\mathcal{P}$ which is defined to be a finite sequence of *instructions* labeled by natural numbers from 1 to some L. The machine operates on two *counters* S_1, S_2 each containing a nonnegative integer. An instruction l is of one of the following three forms with the obvious semantics:

(i) l : `ADD` 1 `TO` S_j; `GOTO` l';

(ii) l : `IF` $S_j \neq 0$ `THEN SUBTRACT` 1 `FROM` S_j; `GOTO` l' `ELSE GOTO` l'';

(iii) l : `STOP`;

where $j \in \{1, 2\}$ and $l, l', l'' \in \{1, \ldots, L\}$.

Without loss of generality we will assume that in every program $\mathcal{P}$ there is a single instruction of type (iii) which is the last one, i.e. its label equals the total number of instructions in the program. The machine starts by executing instruction 1 and works until the command `STOP` is encountered. Note that the execution process is deterministic and has no failure situations. Thus, the execution of a program either ends up with the `STOP` command or lasts forever. It is known [6] that every partial recursive function on natural numbers can be computed by such a program in the following sense:

Theorem 1 (Minsky [6]) *For every partial recursive function f on natural numbers there exists a program $\mathcal{P}$ that applied to the initial counter values $S_1 = 2^d$ and $S_2 = 0$, halts with the counter values $S_1 = 2^{f(d)}$ and $S_2 = 0$ if $f(d)$ is defined, and does not halt otherwise.*

From now on we will always consider programs $\mathcal{P}$ that compute some partial recursive function in the sense of theorem 1 and we will assume that if, starting with counter values $S_1 = 2^d$, $S_2 = 0$, the machine halts, then the final value of S_2 is 0. Theorem 1 implies that it is not generally decidable if for a given d, a program $\mathcal{P}$ terminates on the initial counter values $S_1 = 2^d$, $S_2 = 0$.

Our goal is to prove the following main theorem.

Theorem 2 *Let w_0 be a word with variables over an alphabet A. It is undecidable if given a WRSV $\mathcal{R}$ over A, w_0 is ground reducible by $\mathcal{R}$.*

Proof: Assume that $\mathcal{P}$ is a program for the Minsky machine. The run of $\mathcal{P}$ on an input 2^d can be represented as a sequence of *configurations* (instantaneous descriptions)

$$(1, 2^d, 0), (l_1, s_1^1, s_2^1), \ldots, (l_k, s_1^k, s_2^k), \ldots$$

where a configuration (l_k, s_1^k, s_2^k) means that the machine is about to execute instruction l_k, and s_1^k, s_2^k are the current values of counters S_1 and S_2 respectively. If the execution of $\mathcal{P}$ terminates, the sequence is finite and the final configuration is $(L, 2^m, 0)$ for some $m \geq 0$.

Consider now the alphabet $A = \{*, a, b, \models, \#\}$ and encode a run of the machine by a (finite or infinite) word over A in the following way. A configuration (l_k, s_1^k, s_2^k) is encoded by the word

$$\underbrace{aa\ldots a}_{s_1^k+1}\underbrace{**\ldots *}_{l_k}\underbrace{bb\ldots b}_{s_2^k+1}$$

and the whole run is represented by a sequence of such words separated by $\#$. The first configuration is preceded by the symbols $\models\#$. If the run is finite, the final configuration is followed by $\#\models$ and the coding word has then the following form:

$$\models\#\underbrace{a\ldots a}_{2^d+1}*b\#\ldots\#\underbrace{a\ldots a}_{s_1^k+1}\underbrace{*\ldots *}_{l_k}\underbrace{b\ldots b}_{s_2^k+1}\#\ldots\#\underbrace{a\ldots a}_{2^m+1}\underbrace{*\ldots *}_{L}b\#\models$$

Let $w_0 = \models u \models$ where $u \in \mathcal{X}$, and let $\sigma(w_0)$ be an instance of w_0. We will construct a set of patterns $\mathcal{R}$ which depend on the program $\mathcal{P}$ and on $d \geq 0$ such that if a pattern $p \in \mathcal{R}$ applies to $\sigma(w_0)$, then $\sigma(w_0)$ is not a correct encoding of a (finite) execution of $\mathcal{P}$ on $S_1 = 2^d$, $S_2 = 0$. Moreover, we prove that *every* instance $\sigma(w_0)$ that does not represent a correct finite execution is reducible by some pattern. Thus, $\sigma(w_0)$ is irreducible by $\mathcal{R}$ if and only if $\sigma(w_0)$ encodes the finite execution of $\mathcal{P}$ on $S_1 = 2^d$, $S_2 = 0$ for some $d \geq 0$. Therefore, the set of irreducible instances of w_0 is non-empty iff this set consists of a single instance that represents the finite execution of $\mathcal{P}$ on $S_1 = 2^d$, $S_2 = 0$. Since it is not decidable whether the execution of $\mathcal{P}$ on $S_1 = 2^d$, $S_2 = 0$ is finite or not, we conclude that ground reducibility is not a decidable property.

The patterns of $\mathcal{R}$ given below can be divided into two categories. The first one (groups 1-5) contains patterns without variables. These patterns apply to the instances of w_0 that are not meaningful as they do not follow the syntactical conventions above for representing a run of the Minsky machine. The second category (groups 6-7) contains patterns with variables. Typically, these patterns are designed to apply to a wrong computation step, i.e. a pair of consecutive configurations $\#a^{n+1} *^l b^{m+1}\#a^{n_1+1} *^{l_1} b^{m_1+1}\#$ where the second one does not correctly encode the result of executing command l with $S_1 = n,\ S_2 = m$.

The rest of the proof is devoted to the construction of $\mathcal{R}$. The patterns will be grouped according to the type of "error" in the encoding that they are meant to cover. Each group introduces new patterns and further restricts the set of irreducible instances of w_0. In the following, x, y, z are variables.

1. Rules for $\models$:

$$x \models y \tag{1}$$

   An irreducible string cannot contain $\models$ at an internal position. (Recall that variables are substituted by non-empty words).

$$\models a \tag{2}$$
$$\models b \tag{3}$$
$$\models * \tag{4}$$
$$a \models \tag{5}$$

$$b\models \tag{6}$$
$$*\models \tag{7}$$

$\models$ can be followed and preceded only by $\#$.

2. Rules for syntactical structure:

$$ab \tag{8}$$
$$a\# \tag{9}$$
$$*a \tag{10}$$
$$*\# \tag{11}$$
$$ba \tag{12}$$
$$b\,* \tag{13}$$
$$\#\,* \tag{14}$$
$$\#b \tag{15}$$
$$\#\# \tag{16}$$

It should be clear that the instances of w_0 irreducible by rules 1-16 are exactly the strings described by the following regular expression:

$$\models\#(a^+ *^+ b^+\#)^*\models$$

3. Rules for the initial configuration. These rules apply to all instances of w_0 that do not start with the encoding of the initial configuration, i.e. do not have the prefix $\models\#a^{2^d+1} * b\#$.

$$\models \# \models \tag{17}$$
$$\models \#a^i* \quad \text{for every } i,\ 1\leq i\leq 2^d \tag{18}$$
$$\models \#a^{2^d+2} \tag{19}$$
$$\models \#a^{2^d+1} * * \tag{20}$$
$$\models \#a^{2^d+1} * bb \tag{21}$$

4. Rule for command labels. Suppose that L is the number of commands in $\mathcal{P}$. The pattern
$$*^{L+1} \tag{22}$$
reduces the strings containing more than L consecutive $*$.

5. Rules for the final configuration. Recall that the `STOP` instruction is labeled by L, and S_2 must contain 0 whenever the `STOP` instruction is to be executed. The following two patterns specify that $\models$ can follow none but the final configuration in the encoding.

$$bb\# \models \tag{23}$$
$$a *^i b\# \models \quad \text{for every } i,\ 1\leq i\leq L-1 \tag{24}$$

Conversely, the following pattern applies whenever a final configuration is not followed by $\models$.

$$*^L b\#a \tag{25}$$

The syntactical form of the instances of w_0 irreducible by rules 1-25 is summarized in the following lemma:

Lemma 1 *Let $\sigma(w_0)$ be an instance of w_0 irreducible by rules 1-25. Then $\sigma(w_0)$ has the following general form:*

$$\models\#a^{2^d+1} * b\#a^{p_1} *^{q_1} b^{r_1}\#\dots\#a^{p_i} *^{q_i} b^{r_i}\#\dots\#a^{p_m} *^{q_m} b^{r_m}\#a^{p_{m+1}} *^L b\#\models$$

where $m \geq 0$, $p_i, p_{m+1}, r_i \geq 1$ and $q_i \in \{1,...,L\}$, for every i, $1 \leq i \leq m$.

6. Rules for instructions of type (i). The rules of this group are introduced for each instruction of type (i) in $\mathcal{P}$.

   Assume that l is an instruction of type (i) that applies to S_1. If a string encodes a correct execution and contains a configuration $\#a^{n+1} *^l b^{m+1}\#$ for some $n, m \geq 0$, then it must be followed by the configuration $\#a^{n+2} *^{l'} b^{m+1}\#$. The patterns below are built to be applicable to every substring

   $$...\#a^{n+1} *^l b^{m+1}\#a^{n_1+1} *^{l_1} b^{m_1+1}\#...$$

   where either $l_1 \neq l'$ or $n_1 \neq n+1$ or $m_1 \neq m$.

   Here we face the main difficulty in the proof: if a pattern contains a variable, it can potentially match longer factors, not necessarily restricted to two consecutive configurations, and can possibly apply to some correct executions. The problem is solved by systematically using non-linear variables.

   Let ε denote the empty string. The rules below are schematized using metavariables X, Y, Z where X ranges over $\{\varepsilon, a, aa, aaa, xaax, xaaxa\}$ and Y, Z range over $\{\varepsilon, b, bb, bbb, ybby, ybbyb\}$.

   - handling $l_1 \neq l'$

     $$a *^l bY\#Xa *^{l_1} b \tag{26}$$

     Here l_1 ranges over $\{1, \dots, l'-1\} \cup \{l'+1, \dots, L\}$. Thus, $6 \times 6 \times (L-1)$ patterns are schematized by 26.

   - handling $n_1 \neq n+1$:
     - case $n_1 \leq n$:

       $$Xa *^l bY\#Xa * \tag{27}$$

     - case $n_1 \geq n+2$:

       $$\#Xa *^l bY\#Xaaa \tag{28}$$

- handling $m_1 \neq m$:
  - case $m_1 < m$:
$$a *^l bYZ\#Xa *^{l'} bY\# \tag{29}$$
  - case $m_1 > m$:
$$a *^l bY\#Xa *^{l'} bYb \tag{30}$$

We summarize the effect of the rules of group 6 by the following two lemmas:

Lemma 2 *Let $\sigma(w_0)$ be an instance of w_0 irreducible by rules 1-25. If one of patterns 26-30 applies to $\sigma(w_0)$, then either $\sigma(w_0)$ does not encode a finite execution of the Minsky machine or $\sigma(x) \in a^+$, $\sigma(y) \in b^+$, and $\sigma(z) \in b^+$.*

Proof: We use the fact that in rules 26-30 each variable occurs at a position adjacent to #. Consider, for example, rule 26 and let X be replaced by $xaax$ or $xaaxa$. In either case the pattern contains the substring $\#xaax$. Let σ be the matching substitution. By rules 1-5, $\sigma(x)$ starts with a and ends with either a or #. If $\sigma(x)$ contains at least two #, then it must contain an entire configuration. Since x occurs twice in the pattern, then this configuration must also occur twice in $\sigma(w_0)$. But this implies that $\sigma(w_0)$ does not encode a finite execution. (Recall that the machine works deterministically and a double occurrence of a configuration in the execution implies that the program does not halt.) If $\sigma(x)$ contains exactly one #, then $\sigma(x) = a^p *^q b^r \# a^s$, where $p, q, r \geq 1,\ s \geq 0$. Then the string matched by $\#xaax$ is $\#a^p *^q b^r \# a^{s+2+p} *^q b^r \# a^s$. This string cannot encode an execution step, as counter S_1 is increased by at least 2 which is not possible. (Recall that a command can change a counter at most by 1). Therefore $\sigma(x)$ contains no # and is then composed only of a.

By symmetry, the same reasoning applies to metavariable Y in 26. Moreover, this applies to each variable occurring in a rule, since in every rule each variable has an occurrence adjacent to #. □

Lemma 3 *Let l :* `ADD` 1 `TO` S_j; `GOTO` l' *be a command in $\mathcal{P}$.*

(a) If one of patterns 26-30 applies to an instance $\sigma(w_0)$ irreducible by rules 1-25, then $\sigma(w_0)$ does not encode a correct execution of $\mathcal{P}$.

(b) Conversely, if $\sigma(w_0)$ contains a substring $\#a^{n+1} *^l b^{m+1} \# a^{n_1+1} *^{l_1} b^{m_1+1}\#$ where either $l_1 \neq l'$ or $n_1 \neq n+1$ or $m_1 \neq m$, then one of the rules from 6 is applicable.

Proof: (a) By lemma 2, we can restrict variables x, y, z to be substituted by strings of a^+, b^+, b^+ respectively. Since each of the patterns 26-30 contains the substring $a *^l b$, it can apply only to a substring that correponds to an execution of command l, that is to a substring $\#a^{n+1} *^l b^{m+1} \# a^{n_1+1} *^{l_1} b^{m_1+1}\#$ for some $n, m, n_1, m_1 \geq 0$, $l_1 \in \{0, \ldots, L\}$. It is easy to see that in this case either $l_1 \neq l'$ (if rule 26 applies) or $n_1 \neq n+1$ (if rules 27, 28 apply) or $m_1 \neq m$ (if rules 29, 30 apply). Thus, the matched string cannot be a part of the encoding of a correct execution.

(b) This is proved by simple case analysis on rules 26-30. □

7. If command l of type (i) applies to counter S_2, the rules are constructed in the same fashion and their correctness can be shown with similar arguments.

8. Rules for instructions of type (ii). The rules of this group are introduced for each instruction of type (ii) in $\mathcal{P}$.

   Assume that l is an instruction of type (ii) that applies to S_1. The strings encoding the correct execution are:

$$...\#a^{n+1} *^{l} b^{m} \# a^{n} *^{l'} b^{m} \#...$$

   for $n \geq 1$, and

$$...\#a *^{l} b^{m} \# a *^{l''} b^{m} \#...$$

   otherwise. It is clear that the rules can be built using the same technique as in the previous case. Note that the second string is simpler to treat since the number of a's is fixed.

We summarize the construction of the rules above in the following lemma.

Lemma 4 *Let $d \geq 1$ and $\mathcal{R}$ be the set of all patterns in groups 1-8. For a substitution σ, $\sigma(w_0)$ is reducible by $\mathcal{R}$ iff $\sigma(w_0)$ is not the encoding of a finite execution of $\mathcal{P}$ on $S_1 = 2^d$, $S_2 = 0$.*

Since it is not decidable if the program $\mathcal{P}$ terminates on $S_1 = 2^d$ and $S_2 = 0$, the existence of an irreducible instance of w_0 is not decidable either. This completes the proof of theorem 2. □

It is important to note that the proof could be simplified if we had taken a more complex word w_0 containing, for example, another linear variable. However, we found it interesting to construct a proof for the simplest w_0 possible. Note that the problem is trivially decidable for $w_0 = au$ (symmetrically, for $w_0 = ua$) where $a \in A$, $u \in \mathcal{X}$, by the following observation: w_0 is ground reducible iff for every letter $b \in A$, the word ab is reducible.

3 Related Works

In this paper we have proved the undecidability of the ground reducibility problem for word rewriting systems with variables. From the logical point of view, this problem can be expressed by a positive $\forall\exists$-formula in the first-order theory of a free semigroup. Indeed, let $\bar{x}$ be the variables of a pattern w and $\bar{y}$ be the variables occurring in a rewriting system $\mathcal{R} = \{p_1, \ldots, p_n\}$ over an alphabet A. The ground reducibility of w by $\mathcal{R}$ is equivalent to the validity of the following formula in the free semigroup generated by A:

$$\forall \bar{x} \exists u, z \exists \bar{y} \bigvee_{i=1}^{n} w = u\, p_i\, z$$

Here the number of universal quantifiers is equal to the number of variables in w and the number of existential quantifiers is two more than the maximal number of variables in a rule of $\mathcal{R}$. In particular, the formula corresponding to the proof of theorem 2 contains one universal and five existential quantifiers. In [5] it was proved that the positive first-order $\forall\exists$-theory of free semigroups is in general undecidable. Since we consider a very special form of positive $\forall\exists$-formulae (which, among other restrictions, do not contain conjunction), our result can be regarded as a reinforcement of that of [5] for this particular fragment of the general positive $\forall\exists$-theory of free semigroups. The undecidable theory constructed in [5] has a single universal and four existential quantifiers. The extra existential quantifier that we have in our proof may be considered as the price to pay for restricting the theory.

After this work has been completed we became aware that a result closely related to our theorem 2 has been obtained in [2]. The main difference is that in [2] matching is understood in the usual sense: a pattern matches a string if the latter is an instance of the former. It has been shown in [2] that a single pattern is sufficient in this case to obtain undecidability.

Acknowledgements: We are grateful to Paliath Narendran for his valuable comments.

References

[1] R. V. Book. Thue systems as rewriting systems. *Journal of Symbolic Computation*, 3(1 & 2):39–68, 1987.

[2] T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Inclusion is undecidable for pattern languages. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *Proceedings 20th ICALP Conference, Lund (Sweden)*, volume 700 of *Lecture Notes in Computer Science*, pages 301–312. Springer-Verlag, 1993.

[3] D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient-completeness, quasi-reducibility and their complexity. Technical Report 87-27, Dept. of Computer Science, State University of New York at Albany, 1987.

[4] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[5] S.S. Marchenko. Undecidability of the positive $\forall\exists$-theory of a free semigroup. *Sibirskii Matematicheskii Zhurnal*, 23(1):196–198, 1982. in Russian.

[6] M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.

[7] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[8] A. Salomaa. Pattern Languages: Problems of Decidability and Generation. In Z. Ésik, editor, *Proceedings 9th FCT Conference*, volume 710 of *Lecture Notes in Computer Science*, pages 121–132, Szeged (Hungary), 1993. Springer-Verlag.