Information and Computation **174**, 1–32 (2002)
doi:10.1006/inco.2002.3151

The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems

Sergei Vorobyov[1,2]

Department of Information Technology Uppsala University, Box 337, 751 05 Uppsala Sweden
E-mail: vorobyov@csd.uu.se
URL: www.csd.uu.se/∼vorobyov

Received May 28, 1998; revised September 13, 2000

By reduction from the halting problem for Minsky's two-register machines we prove that there is no algorithm capable of deciding the $\exists\forall\forall$-theory of one step rewriting of an arbitrary *finite linear confluent finitely terminating* term rewriting system (weak undecidability). We also present a *fixed* such system with undecidable $\exists\forall^*$-theory of one step rewriting (strong undecidability). This improves over all previously known results of the same kind. © 2002 Elsevier Science (USA)

Key Words: term rewriting system; confluence; termination; linearity; decidability; first-order theory of one step rewriting; quantifier prefix classes.

1. INTRODUCTION

A finite term rewriting system R generates the binary *one step reducibility relation* **R** on the set of ground terms. A *theory of one step rewriting* in R is the first-order theory of this binary relation **R** formulated in the language of the predicate calculus *without equality* containing the *unique binary predicate* symbol R interpreted as **R**. The problem whether first-order theories of one step rewriting in finite systems are decidable was suggested in [1, p. 331] and repeated in the *Rewriting Techniques and Applications* (*RTA*) lists of open problems; see [2, p. 473] and [3, p. 461].

The motivation for the problem is quite natural. For example, the *ground reducibility* of a term $t(\bar{x})$ and the *strong confluence* of a system are expressible by the formulas $\forall\bar{x}\exists y R(t(\bar{x}), y)$ and $\forall x, y, z\exists w(R(x, y) \wedge R(x, z) \Rightarrow R(y, w) \wedge R(z, w))$, respectively. Note that both properties are known to be decidable. Similarly, the decidability of properties like *encompassment*, known to be decidable [1, 4], would follow from the general decidability of theories of one step rewriting. Recall also that the first-order theories of one step rewriting in *finite ground* systems are decidable [5]. On the other hand, the transitive closure of the one step reducibility relation seems to be inexpressible in the theories of one step rewriting (the opposite would immediately lead to their undecidability). All these facts motivated the quest for the solution to the above problem and for the general decision procedure applicable to all rewrite systems. This would have allowed us to decide all properties of rewrite systems, like discussed above, expressible in the language of one step rewriting *uniformly*.

Unfortunately, the problem was settled in the negative (undecidable). It was demonstrated in [6], by reduction from the post correspondence problem, that there is no algorithm capable of deciding the $\exists\exists\forall$-theory of an arbitrary term rewriting system. This result, however, does not imply the existence of any fixed rewrite systems with undecidable theories. Moreover, each particular rewrite system has a decidable $\exists\exists\forall$-fragment (used in [6]). Actually, this holds for any other fragment with finitely many quantifier prefixes, as explained in Section 5. On the other hand, [7] presented a simple *fixed* rewrite rule system with undecidable theory of one step rewriting, by using a reduction from the undecidable theory of binary concatenation (or free semigroups); see [8]. We therefore distinguish between the *weak*

[1] To whom correspondence should be addressed. Fax: +46-18-51-19-25.

[2] Preliminary version with weaker results based on different ideas and proofs appeared in the Proceedings of the 8th International Conference on Rewriting Techniques and Applications (RTA'97), June 2–4, 1997, Sitges (Barcelona), Spain, Lecture Notes in Computer Science, Vol. 1232, pp. 254–268. Preliminary version of this publication appeared as Research Report MPI-I-98-009, Max-Planck Institut für Informatik, Saarbrücken, Germany, May 1998, `http://data.mpi-sb.mpg.de/internet/reports.nsf/NumberView/`.

0890-5401/02 $35.00

undecidability, i.e., non-existence of a unique decision algorithm applicable to all systems uniformly, and *strong undecidability*, i.e., undecidability of the theories of fixed systems.

It should be noted that both [6] and [7] constructed non-finitely terminating and non-linear[3] rewrite rules.[4] Moreover, [6] directly used the rules of the form $t \to t$ one hardly ever encounters in practice. This somehow diminished the relevance of the obtained results and left a strong hope that the theories of one step rewriting should be decidable for finitely terminating systems.

In these circumstances H. Ganzinger at RTA'96 (New Brunswick, NJ) suggested a problem whether finite finitely terminating systems have (un)decidable theories of one step rewriting. Recall in this connection that the confluence is undecidable, in general, but becomes decidable for finite finitely terminating systems. The similar decidability problem was put forward for the subclass of linear systems.

The decidability conjecture was first dispelled in [10], where a fixed finite, simultaneously finitely terminating and linear system with undecidable theory of one step rewriting was constructed. The proof again was given by reduction from the undecidable theory of binary concatenation. As a partial drawback compensating for the ease of reduction, the quantifier alternation of the sentences forming the undecidable class was quite high. Then in [11] it was shown that no algorithm is capable of deciding the $\exists^*\forall^*$-theory of one step rewriting of an arbitrary finite finitely terminating system (but again without implying undecidability for any fixed systems; see Section 5). A similar result for terminating right-ground but non-linear systems is also proved in [11].

In this paper we further improve and sharpen the above *weak* undecidability results by showing that no decision algorithm can decide the $\exists\forall\forall$-theory of any given finite, simultaneously (1) finitely terminating, (2) linear, and (3) confluent rewrite system. All the preceding proofs constructed *non-confluent* systems. For comparison, [11] proved an analogous result for non-confluent terminating systems and $\exists\exists\forall\forall\forall\forall$-theories, and [6] for divergent non-confluent systems.

We also prove *strong undecidability* by constructing a *fixed* finite linear canonical system with undecidable $\exists\forall^*$-theory of one step rewriting. Recall that the weak undecidability results of [6] and [11] do not imply existence of such systems (Section 5), whereas [7, 10] used much more complicated quantifier prefixes and *non-confluent* systems. As a methodological advantage of the proof presented here let us mention the use of reduction from the well-known undecidable halting problem for the two-counter machines [12–14]. Note that [11] used a rather complicated specially tailored undecidability problem in his proof (the details have not yet been published).

The main results of the paper are summarized in the following:

Main Theorem

(Part A: Weak undecidability). There is no algorithm deciding the $\exists\forall\forall\forall$-theory of one step rewriting for every finite linear canonical system.

(Part B: Strong undecidability). There exists a finite linear canonical rewrite system (explicitly presented) with undecidable (actually, r.e.-complete) $\exists\forall^*$-theory of one step rewriting.

Note that Part A refers to a uniform algorithm that first reads a system R as a parameter and then adjusts itself to decide its theory $Th_{\exists\forall\forall\forall}(\mathrm{R})$.

We call Part A "weak undecidability" for three reasons:

(1) it has logical form $\neg\exists A\forall\mathrm{R}$, weaker compared with $\exists\mathrm{R}\forall A\neg$ of strong undecidability,

(2) it does not imply strong undecidability (see Section 5),

(3) for every finite term rewriting system and for every finite quantifier prefix like $\exists\forall\forall\forall$, $\exists\exists\forall$, $\exists\exists\forall\forall\forall\forall$ (but not for $\exists\forall^*$, which denotes an infinite set of quantifier prefixes) the corresponding theory of one step rewriting with this finite prefix is *always decidable* (see Section 5). This, somehow, diminishes the practical value of Treinen–Marcinkowski's results. Indeed, one almost never deals with all rewrite systems altogether, but rather with one fixed given system at a time. But for any fixed system and any finite quantifier prefix Q, the Q-theory of the system is always decidable. Thus weak undecidability is immaterial for practical purposes.

[3] i.e., containing repeated variable occurrences on the left (or right) hand side.

[4] Later this was improved in [9] to linear shallow systems, but still non-terminating with rules $t \to t$.

Outline. The paper is organized as follows. After preliminaries in Sections 2–4, in Section 5 we discuss and relate *weak* and *strong* undecidability. Section 7 introduces Minsky's two register machines, and Section 8 describes the idea of reduction from the halting problem for these machines, which we employ in the proof. Sections 9–16 implement the reduction. Sections 17 and 18 summarize all rewrite rules and formulas constructed. Section 19 is devoted to the correctness proof. Section 20 proves undecidability of the $\exists\forall\forall\forall$-theories for finite right-ground canonical systems, which improves (simpler prefix, confluent systems) over [11]. In Section 21 we prove *strong undecidability* for $\exists\forall^*$-theories of fixed linear canonical systems. Finally, in Section 22 we show strong undecidability for $\exists\forall\forall\forall$-theories, when function symbols are allowed in formulas. We conclude in Section 23.

2. PRELIMINARIES

We suppose familiar use throughout the standard basic notions of term rewriting; see, e.g., [15, 16]. Specifically, by $r[t]$ we denote a term r containing a distinguished occurrence of a subterm t. By $r[s/t]$ we denote the result of replacing this distinguished occurrence with term s. We freely speak about reducibility in the outermost and inner positions, etc. We also expect some knowledge of finite termination and the Knuth-Bendix critical pairs algorithm; see, e.g., [15–17].

A rewrite system is *canonical* if it is simultaneously finitely terminating and confluent. A system is *linear* if each term in its left- and right-hand sides is linear, i.e., contains at most one occurrence of every variable.

In writing predicate formulas we omit parentheses assuming the usual priority precedence of boolean connectives: $\neg$, $\wedge$, $\vee$, $\Rightarrow$.

3. THEORY OF ONE STEP REWRITING

Given a functional signature Σ with constants and a finite rewrite rule system R, consider the rewrite model $M = \langle T(\Sigma), \mathbf{R}\rangle$ induced by R, where $T(\Sigma)$ is the Herbrand universe over Σ and the relation

$$\mathbf{R} = \{\langle s, t\rangle \mid s, t \in T(\Sigma) \wedge s \to_R t\} \subseteq T(\Sigma) \times T(\Sigma)$$

is the one step rewrite relation on $T(\Sigma)$ generated by the system R.

Let $\mathcal{L}$ be the first-order language without equality containing the only binary predicate symbol R. The first-order theory of one step rewriting in R is the set of sentences of $\mathcal{L}$ true in the rewrite model M, when the binary predicate symbol R is interpreted as the binary relation $\mathbf{R}$. This theory is denoted *Th*(R).

Remark 1. It is important to note that the only non-logical symbol used in formulas of the theory is R, and the functional symbols of signature Σ are not allowed in formulas.[5] Sometimes instead of strict notation $R(x, y)$ for atomic formulas of the theory we use more familiar and intuitive notation $x \to y$ (not to be confused with rewrite rules).

Remark 2. One can easily construct an infinite system with the undecidable existential theory of one step rewriting. It suffices to represent the addition and multiplication tables by rewrite rules and use Matiyasevich's result on undecidability of Diophantine equations.

More explicitly, consider the system with the following rules for every natural $m, n > 2$, where p, q, m, a are auxiliary binary symbols and $\underline{x \cdot y}$, $\underline{x + y}$ denote the numerals equal to the product $x \cdot y$ and the sum $x + y$:

$$\begin{aligned}
p(x, y) &\to p(x, y),\\
p(x, y) &\to q(x, y),\\
q(x, y) &\to p(x, y),\\
q(x, y) &\to x,\\
p(x, y) &\to y,
\end{aligned}$$

[5] We will relax this restriction in Section 22.

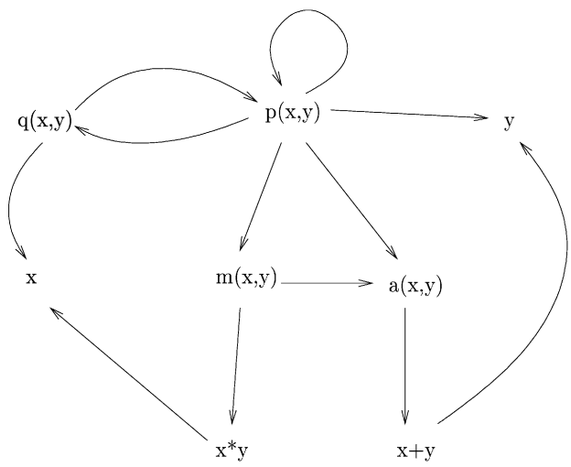


FIG. 1. Possible rewrites in the system of Remark 2.

$$
\begin{aligned}
p(x, y) &\to m(x, y),\\
p(x, y) &\to a(x, y),\\
m(x, y) &\to a(x, y),\\
m(x, y) &\to \underline{x \cdot y},\\
a(x, y) &\to \underline{x + y},\\
\underline{x \cdot y} &\to x,\\
\underline{x + y} &\to y.
\end{aligned}
$$

Figure 1 shows the diagram of possible rewrites.

Now, instead of saying $\exists u, v \cdots \wedge u \cdot v = w \wedge \ldots$ (while expressing the existence of solutions to a Diophantine equation; treating of $u + v = w$ is completely analogous), one writes:

$$
\begin{aligned}
&\exists P, Q, A, M, U, V, X_{U\cdot V}, Y_{U+V} \ldots\\
&\quad P \to P \wedge P \to Q \wedge Q \to P \wedge Q \not\to Q \wedge Q \to U \wedge U \not\to P \wedge\\
&\quad P \to V \wedge V \not\to V \wedge P \to M \wedge M \not\to M \wedge P \to A \wedge A \not\to A \wedge\\
&\quad M \to A \wedge M \to X_{U\cdot V} \wedge X_{U\cdot V} \to U \wedge A \to Y_{U+V} \wedge Y_{U+V} \to V \ldots,
\end{aligned}
$$

meaning that P is a pair of U, V, and $X_{U\cdot V}$, Y_{U+V} (just variables with fancy subscripts) equal the product and the sum of U and V, respectively.

The assumption $m, n > 2$ is made to keep the system reducing, except for the first two rules needed to say "P is a pair." Anyway, the Diophantine equations problem remains undecidable in natural numbers $> k$ (for any natural k). The reader will readily fill out the details and find possible simplifications after reading the paper.

4. THEORIES OF ONE STEP REWRITING WITH RESTRICTIONS ON QUANTIFIER PREFIXES

It is well known that each first-order sentence is equivalent to a sentence in the *prenex form*

$$Q_1 x_1 \ldots Q_n x_n \Phi,$$

where $Q_i \in \{\exists, \forall\}$ are quantifiers and Φ is a quantifier-free formula.

A *quantifier prefix type* is a regular expression over the alphabet $\{\exists, \forall\}$; for example, $\exists\forall\forall$, $\exists^*\forall^*$, $\exists\forall \cup \forall\exists$. Given a quantifier prefix type $\mathbf{Q}$, let $L(\mathbf{Q})$ be the language defined by the regular expression $\mathbf{Q}$ according to the usual rules. This language may be finite, as in the case of $\mathbf{Q} = \exists\forall\forall$ (one element), or infinite, as in the case of $\mathbf{Q} = \exists^*\forall^*$.

For a given quantifier prefix type $\mathbf{Q}$, the $\mathbf{Q}$-*theory of one step rewriting in* R is a subset of *Th*(R) consisting of prenex sentences with quantifier prefixes in $L(\mathbf{Q})$. This theory is denoted by $Th_{\mathbf{Q}}(\mathrm{R})$.

In the first part of this paper we will prove weak undecidability of $\exists\forall\forall$-theories of one step rewriting in linear canonical systems. For comparison, [11] proved weak undecidability of $\exists\exists\forall\forall\forall\forall$-theories for linear terminating non-confluent systems, and [6] proved weak undecidability of $\exists\exists\forall$-theories of one step rewriting in non-terminating non-linear systems. In the second part of the paper, in Section 21, we prove strong undecidability of the $\exists\forall^*$-theory of a particular system.

5. WEAK VS. STRONG UNDECIDABILITY RESULTS

The results of [6, 11, 10] are sometimes misinterpreted or misunderstood, and some clarification is necessary.

Let us first recall the statement of the problem, as given in the RTA'93, RTA'95 lists of open problems; see [2, 3].

Problem 5.1 (RTA'93, RTA'95). For an arbitrary finite term rewriting system R, is the first-order theory of one step rewriting $\rightarrow_{\mathrm{R}}$ decidable? . . .

This informal statement allows for at least two different interpretations, depending on the order of quantification (note that $\neg\,(2) \Rightarrow \neg\,(1)$):

Problem 5.1 (Formalized). Prove or disprove that:

$$\exists \ \textit{an algorithm } A \ \forall \ \textit{system } \mathrm{R} \ (A \textit{ decides } Th(\mathrm{R})), \tag{1}$$

$$\forall \ \textit{system } \mathrm{R} \ \exists \ \textit{an algorithm } A \ (A \textit{ decides } Th(\mathrm{R})). \tag{2}$$

Thus, [6, 11] disproved (1) by showing

(Weak undecidability). There is no algorithm that given a finite term rewriting system R decides its theory *Th*(R) of one step rewriting. Even stronger, there is no algorithm that:

(1) given a finite (but otherwise unrestricted) rewrite system R decides its $\exists\exists\forall$-theory of one step rewriting $Th_{\exists\exists\forall}(\mathrm{R})$, [6];

(2) given a finite linear finitely terminating system R decides its $\exists\exists\forall\forall\forall\forall\forall$-theory of one step rewriting $Th_{\exists\exists\forall\forall\forall\forall\forall}(\mathrm{R})$, [11].

This settles Problem 5.1 in the form (1) in the negative.

However, it might happen (see below) that simultaneously one has

(Non-uniform decidability). For each finite rewrite rule system R_i the corresponding first-order theory $Th(\mathrm{R}_i)$ of one step rewriting is *decidable* by some (non-uniform) algorithm A_i.

And in this latter case one should admit that Problem 5.1 is settled *in the positive*, because it corresponds more exactly (at least from the author's point of view) to what is asked for in the statement of Problem 5.1.

Although the results of [7, 10], exclude non-uniform decidability by disproving (2), the results of [6] and [11] do not exclude it. This follows from the fact that both authors use only finite quantifier prefixes and from the next easy

PROPOSITION 3. *For every finite rewrite rule system and every quantifier prefix type* $\mathbf{Q}$ *describing a finite language* $L(\mathbf{Q})$, *the* $\mathbf{Q}$*-theory of one step rewriting is decidable. In particular*, $\exists\exists\forall$- *and* $\exists\exists\forall\forall\forall\forall\forall$-*theories of one step rewriting are decidable in every fixed finite rewrite system.*

Proof. Given a finite quantifier prefix $Q_1 \dots Q_n$, the language $\mathcal{L}$ of the theory of one step rewriting has (see Section 3):

(1) only finitely many different atoms with variables in $\{x_1, \dots, x_n\}$ (since there are no function symbols in $\mathcal{L}$);

(2) only finitely many literals and non-equivalent quantifier-free boolean formulas with variables in $\{x_1, \dots, x_n\}$;

(3) consequently, only finitely many non-equivalent sentences with quantifier prefix $Q_1 \dots Q_n$.

Therefore, the $Q_1 \dots Q_n$-theory contains only finitely many equivalence classes of sentences and consequently is decidable, because every finite set is always decidable. ■

Although for a given finite rewrite rule system R_i and a prefix $Q_1 \dots Q_n$ the corresponding individual decision algorithm may be quite sophisticated, it *always exists*. One cannot collect all such algorithms (parameterized by a system) in just one generic algorithm, because this would contradict the (weak undecidability) proved in [6, 11].

On the other hand, [7, 10] showed

(*Strong undecidability*). There exist finite term rewriting systems with undecidable theories of one step rewriting.

This settles Problem 5.1 in the form (2) in the negative.

6. OUTLINE OF THE PAPER

In the first part of the paper (until Section 21) we improve the result of [11] on weak undecidability by proving

THEOREM A (Weak undecidability of $\exists\forall\forall\forall$-theories for linear canonical systems). *There is no decision algorithm that given a finite linear canonical term rewriting system decides its $\exists\forall\forall\forall$-theory of one step rewriting.*

For comparison, [11] proved weak undecidability of the $\exists\exists\forall\forall\forall\forall$-theories, for linear terminating non-confluent systems. Hence our result gives an improvement both in terms of a simpler prefix and interms of a more restrictive class of rewrite rules.

Theorem A establishes the strongest currently known weak undecidability result for the theories of one step rewriting in Noetherian systems.

In the second part of the paper (Section 21) we improve the results of [7, 10] on strong undecidability by proving.

THEOREM B (Strong undecidability of $\exists\forall^*$-theories for linear canonical systems). *There exists* (*and can be explicitly presented*) *a finite linear canonical term rewriting system with undecidable $\exists\forall^*$-theory of one step rewriting.*

For comparison, [6] proved weak undecidability for $\exists\exists\forall$-theories in non-terminating, non-linear, non-confluent systems, and [9] proved weak undecidability for $\exists\exists\forall$-theories in non-terminating (with rules $t \to t$) non-confluent but linear and shallow systems. Strong undecidability proofs appeared only in [7, 10].

7. MINSKY'S TWO-REGISTER MACHINE

Our undecidability proof is by reduction from the well-known *halting problem for the two-register machine* (2RM) [12–14] In the definition below we make several simplifying technical assumptions discussed later in Remark 6.

DEFINITION 4 (2RM). A 2RM is an automaton with a finite program and two unbounded counters (called the *left* and the *right* registers) capable of storing arbitrary natural numbers. A 2RM-program P

is a finite list of consecutively labeled commands

$$1 : Command_1;\ \ldots;\ p : Command_p,$$

where $p \geq 2$ is the number of commands in P and each $Command_i$ is of one of the following five kinds:

Halt. By executing this command the 2RM halts. We assume that the last command in a program is always Halt, and this is the unique Halt command in a program.

Add 1 to the left register. By executing the command $i : AL$ the 2RM increases the contents of the first (left) register by one, leaves the second (right) register unchanged, and proceeds to the next command $i + 1$. We assume that the first command in a program is always $1 : AL$.

Add 1 to the right register. By executing the command $i : AR$ the 2RM increases the contents of the second (right) register by one, leaves the first (left) register unchanged, and proceeds to the next command $i + 1$.

Subtract 1 from the left register. By executing the command $i : SL, j$ the 2RM does the following:

- if the contents of the first (left) register are *positive*, the 2RM decreases it by one, leaves the second (right) register unchanged, and proceeds to the command labeled j, where $2 \leq j \leq p$;
- otherwise, if the contents of the first (left) register are *zero*, the 2RM leaves both registers unchanged and proceeds to the next command $i + 1$.

Subtract 1 from the right register. The execution of $i : SR, j$ is analogous to those of $i : SL, j$, with the roles of the left and the right registers interchanged.

The 2RM-halting problem is undecidable [12–14]. More precisely:

THEOREM 5 (Inputless version). *It is undecidable, given a program P for the 2RM, to say whether the machine halts when started with the first instruction of P and both registers containing zeros.*

We will also make use of a version of this theorem for the 2RM with input (see Theorem 21) to prove strong decidability of the $\exists\forall^*$-theories of one step rewriting in Section 21.

We finish this section by giving explanations concerning the technical assumptions in Definition 7.

Remark 6.

(1) We assume that the number p of commands in a 2RM program is greater than one, since for the (unique) one-command program $1 :$ Halt termination is immediate.

(2) By always starting a program with $1 : AL;\ 2 : SL, 3$ we may assume that every program starts with $1 : AL$ and the control never returns to the command labeled 1. Indeed, given a program P we can write $1 : AL;\ 2 : SL, 3$ in front of it and then systematically change labels (by adding 2 to each one) in P. The modified program halts iff the initial does. The role of these technical assumptions will become clear later, in Sections 13, 19.5.

8. REDUCTION: PROOF IDEA

In the first part of the paper, until Section 21, we will:

(1) present a fixed $\exists\forall\forall\forall$-sentence (4), independent of a rewrite rule system, and

(2) show how, given a 2RM program P, to effectively construct a finite linear canonical system R

such that the sentence (4) below is true in the theory *Th*(R) of one step rewriting in R if and only if the 2RM executing P halts after a finite number of steps. Together with Theorem 5 this will immediately imply Part A of our main theorem on weak undecidability.

In Section 21 we will show how to obtain fixed explicit examples of finite linear canonical rewrite systems with undecidable $\exists\forall^*$-theories of one step rewriting. The 2RM will be modified to accept inputs: in the initial state both registers will contain a natural number n, the program P will be fixed, but

the $\exists\forall^*$-sentences H_n expressing halting of the 2RM with input n will vary and form the undecidable theory. This will prove Part B of the main theorem on strong undecidability.

9. SENTENCE EXPRESSING HALTING

A run of the 2RM executing a program P is a finite sequence of instantaneous descriptions (IDs) represented by triples of natural numbers

$$\langle x_0, y_0, z_0\rangle, \ldots, \langle x_m, y_m, z_m\rangle, \quad (m \geq 1) \tag{3}$$

where x_i's are the left register contents, y_i's are the right register contents, and z_i's are command labels. The intuitive interpretation is that $\langle x_{i+1}, y_{i+1}, z_{i+1}\rangle$ is obtained from $\langle x_i, y_i, z_i\rangle$ as a result of execution of the z_ith command of P with the left- and right-register contents equal to x_i and y_i respectively, as defined in Section 7. The *initial ID* $\langle x_0, y_0, z_0\rangle$ is $\langle 0, 0, 1\rangle$ and in the *final ID* $z_m = p$ (recall that p is the number of commands in P). The formal definition of a run is straightforward from Definition 4 and we omit it here.

To prove Part A of the main theorem, we will write a fixed sentence, independent of a program P, expressing that the 2RM executing P halts starting in the initial ID $\langle 0, 0, 1\rangle$. This sentence will be written in the form

$$H \equiv_{df} \exists r(C_1(r) \wedge C_2(r) \wedge C_3(r) \wedge E(r)), \tag{4}$$

where $C_{1,2,3}(r)$ and $E(r)$ are formally defined below in such a way that:

- $\exists r$ says "there exists a *run* r,"
- $C_1(r) \wedge C_2(r) \wedge C_3(r)$ says that r is a structurally quasi-correct[6] (see Sections 13, 14, 15) sequence of IDs of the 2RM executing a program P, and the control flow in r is correct[7] according to Definition 4,
- $E(r)$ says that the registers are operated correctly[8] along the run r, according to Definition 4, and r starts with the initial ID $\langle 0, 0, 1\rangle$.

Thus the whole sentence (4) says that there exists a finite correct terminating *run* r of the 2RM executing the program P.

10. HOW TO TRANSLATE MACHINE COMMANDS

Our aim in this section is to describe the intuition for writing the most sophisticated part $E(r)$ of the sentence (4) and the corresponding part of the rewrite rule system.

Suppose we have a *run candidate*, i.e., a sequence r of the form (3) (in list representation described below), in which the flow of control is *correct*. The latter means, informally, that z_i's in r follow correctly; e.g., if $i : AL$ is in P then $\langle \ldots, i\rangle, \langle \ldots, j\rangle$ with $j \neq i + 1$ does not appear in r. Such a correctness will be guaranteed by the part $C_1(r)$ of (4) (described in Sections 13, 14, 15) occurring conjunctively with $E(r)$ in (4). So, assuming this control flow correctness, we need to check, by using linear canonical rules, whether the contents of registers are modified correctly along a run candidate r.

The main idea is to construct rewrite rules in a way to simultaneously satisfy the following two properties:

(1) every adjacent pair of triples $\langle x_i, y_i, z_i\rangle, \langle x_{i+1}, y_{i+1}, z_{i+1}\rangle$ in a sequence (3) representing a run candidate r could be reduced to form the following rewrite diagram (no matter whether the transition

[6] For example, does not contain 'senseless' things like $\langle\langle \ldots, \ldots, \ldots\rangle, \ldots, \ldots\rangle$.
[7] For example, if P contains $9 : AL$ then a run does not contain adjacent triples like $\langle x, y, 9\rangle, \langle u, v, 8\rangle$.
[8] For example, if $7 : AL$ is in P and a run contains the adjacent pair of triples $\langle x, y, 7\rangle, \langle u, v, 8\rangle$ then $u = x + 1$ and $v = y$.

from the ID $\langle x_i, y_i, z_i\rangle$ to the ID $\langle x_{i+1}, y_{i+1}, z_{i+1}\rangle$ is correct or not):

$$\begin{array}{ccc} \langle x_i, y_i, z_i\rangle, \langle x_{i+1}, y_{i+1}, z_{i+1}\rangle & \rightarrow & w_2 \\ \downarrow & & \downarrow \\ w_0 & \leftarrow & w_1 \end{array} \tag{5}$$

for some w_0, w_1, w_2, and, moreover,

(2) the diagram (5) can be *completed* by the $\swarrow$ rewrite to the diagram

$$\begin{array}{ccc} \langle x_i, y_i, z_i\rangle, \langle x_{i+1}, y_{i+1}, z_{i+1}\rangle & \rightarrow & w_2 \\ \downarrow \swarrow & & \downarrow \\ w_0 & \leftarrow & w_1 \end{array} \tag{6}$$

if and only if the register contents are operated correctly in the transition from $\langle x_i, y_i, z_i\rangle$ to $\langle x_{i+1}, y_{i+1}, z_{i+1}\rangle$.

Therefore, the part $E(r)$ of (4) can be expressed by the $\forall\forall\forall$-formula

$$E(r) \equiv_{df} \forall w_0, w_1, w_2 (R(r, w_0) \wedge R(r, w_2) \wedge R(w_2, w_1) \wedge R(w_1, w_0) \Rightarrow R(w_2, w_0)). \tag{7}$$

This idea is implemented in Section 12.

The formula (7) looks more intuitive when written in the form

$$\forall w_0, w_1, w_2 \left(\begin{array}{ccc} r & \rightarrow & w_2 \\ \downarrow & & \downarrow \\ w_0 & \leftarrow & w_1 \end{array} \Rightarrow \begin{array}{cc} & w_2 \\ \swarrow & \\ w_0 & \end{array} \right).$$

11. SIGNATURE AND NOTATIONAL CONVENTIONS

The signature Σ we will use in constructing rewrite systems and formulas is as follows:

- a constant ε to represent the empty list;
- a constant 0 to represent the natural number zero;
- binary function $c(,)$ for the list constructor;
- unary $s(\,)$ for the successor on natural numbers;
- ternary $\langle , , \rangle$ for the triple constructor;
- constants a, b, d, auxiliary;
- binary functions h, f, auxiliary.

CONVENTION 1. *In the following we will formally represent the run sequence* (3) *as a term* (*list*)

$$[\langle x_0, y_0, z_0\rangle, \ldots, \langle x_m, y_m, z_m\rangle], \quad (m \geq 1) \tag{8}$$

where, as usual, $[\,] = \varepsilon$ *and* $[e_0, e_1, \ldots, e_n] = c(e_0, [e_1, \ldots, e_n])$, *with the constant* ε *for the empty list and the binary list constructor* $c(,)$. *Thus,* (8) *is a* right-flattened *list of triples of natural numbers built from the empty list* ε *by using the binary list constructor. Below we will freely switch between the informal representation of a run* (3) *and its formal list representation* (8), *keeping in mind that the relation between them is obvious.*

CONVENTION 2. *Formally, a sequence of the form* (3) *is represented by a right-flattened list* (8) *of triples built using the list constructor c. Sometimes, to simplify readability we present rewrite rules in the form* $[\langle \ldots\rangle, \langle \ldots\rangle \ldots] \rightarrow t$ *or* $\langle \ldots\rangle, \langle \ldots\rangle \rightarrow t$, *instead of the less readable* $c(\langle \ldots\rangle, c(\langle \ldots\rangle, u) \rightarrow t$ (*where u is a fresh variable*). *It will always be clear how to transform this shorthand into a formal long form.*

CONVENTION 3. *To improve readability we will often depict rewrite rules $l \to r$ in a slightly unconventional way, with arrows going in different directions, as in rules* (⇓), (↙), (9) *below.*

CONVENTION 4. *In the rules and formulas we write below x, y, z, u, v, w are variables, while i, j, k, l, m, n are natural numbers. For a natural number i, $\underline{i}$ denotes the term $s^i(0)$. Sometimes, when it does not lead to confusion, we use the usual decimal numbers instead of the formal numerals $s^i(0)$ in unary notation. In writing terms with unary function symbols we usually omit parentheses.*

12. TRANSLATING COMMANDS INTO REWRITE RULES

Assume that P is an arbitrary *but* fixed 2RM program with $p \geq 2$ commands, starting with $1 : AL$. We proceed to compiling P into a system of linear canonical rewrite rules R. Thus the system R depends on a program P, i.e., $\mathrm{R} \equiv \mathrm{R}(P)$; see Section 8.

12.1. Auxiliary (⇓) Rule

The following rule will be used to commute rewrite diagrams created by other rewrite rules, with the intention to check properties of terms (as we described in Section 10).

$$\begin{array}{c} h(u, v) \\ \downarrow \\ f(u, v) \end{array} \qquad (\Downarrow)$$

12.2. Shortcut Rules ($\swarrow_{1,2}$)

The following two rules will also be used to commute rewrite diagrams (cf. (5), (6) above) created by other rewrite rules on terms satisfying certain properties:

$$\begin{array}{rl} & [h(\langle 0, 0, s0\rangle, \langle 1, 0, v\rangle), \dots] \\ \swarrow & \\ [\langle 0, 0, s0\rangle, 0, \langle 1, 0, v\rangle, 0, \dots] & \end{array} \qquad (\swarrow_1)$$

$$\begin{array}{rl} & [u, h(\langle x', y', ssz\rangle, \langle x, y, v\rangle), \dots] \\ \swarrow & \\ [u, \langle x', y', ssz\rangle, 0, \langle x, y, v\rangle, 0, \dots] & \end{array} \qquad (\swarrow_2)$$

These rules are, of course, more readable versions of the following two rules

$$\begin{array}{rl} & c(h(\langle 0, 0, s0\rangle, \langle 1, 0, v\rangle), w) \\ \swarrow & \\ c(\langle 0, 0, s0\rangle, c(0, c(\langle 1, 0, v\rangle, c(0, w)))) & \end{array}$$

$$\begin{array}{rl} & c(u, c(h(\langle x', y', ssz\rangle, \langle x, y, v\rangle), w)) \\ \swarrow & \\ c(u, c(\langle x', y', ssz\rangle, c(0, c(\langle x, y, v\rangle, c(0, w))))) & \end{array}$$

respectively, according to our Conventions 1, 2 on lists.

Remark 7. The difference between ($\swarrow_1$) and ($\swarrow_2$) is crucial for our purposes: pay attention to $s0$ in the rule ($\swarrow_1$) and ssz in the rule ($\swarrow_2$). Note that we do not introduce just one generic rule

$$\begin{array}{rl} & c(h(\langle x', y', sz\rangle, \langle x, y, v\rangle), w) \\ \swarrow & \\ c(\langle x', y', sz\rangle, c(0, c(\langle x, y, v\rangle, c(0, w)))) & \end{array}$$

instead of ($\swarrow_1$) and ($\swarrow_2$). The reason is that we wish to distinguish between the cases for 'one' ($s0$) and 'greater than one' (ssz). Note that ($\swarrow_1$) applies in the *head* of a list, whereas ($\swarrow_2$) applies in the

tail (second element) of a list. This complication is needed to ensure that a run r witnessing the validity of (4) starts with the *initial ID* $\langle 0, 0, 1\rangle$, i.e., has form $c(\langle 0, 0, 1\rangle, \ldots)$; see below Section 19.5. Note also that the form of the rule $(\swarrow_1)$ assumes that the first command in a program is always $1 : AL$; see Remark 6.

CONVENTION 5. *In all the rules and diagrams below the effect of commutation by* $(\Downarrow)$, $(\swarrow_{1,2})$ *will be depicted as* $\Downarrow$, $\swarrow$ *respectively. In these contexts* $\Downarrow$, $\swarrow$ do not define *new rewrite rules*, *but denote rewrite steps made by* $(\Downarrow)$, $(\swarrow_{1,2})$ *and are added as comments to clarify intuition.*

12.3. Addition Commands

12.3.1. *Left Addition Command*

The command $i : AL$ is translated into three linear rewrite rules, $\rightarrow$, $\downarrow$, and $\leftarrow$ given below (recall that $\Downarrow$ is not a rule, but a rewrite step made by the rule $(\Downarrow)$ given above):

$$\begin{array}{rcl} c(\langle x, y, \underline{i}\rangle, c(\langle s(u), v, z\rangle, w)) & \rightarrow & c(h(\langle u, v, i\rangle, \langle s(x), y, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x, y, \underline{i}\rangle, c(0, c(\langle s(u), v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, v, i\rangle, \langle s(x), y, z\rangle), w) \end{array} \tag{9}$$

Note that the $\rightarrow\Downarrow\leftarrow$ combination in diagram (9) makes two swaps of variables: $x, y, u, v \mapsto u, v, x, y \mapsto x, y, u, v$. Along both $\downarrow$ and $\rightarrow\Downarrow\leftarrow$ paths in (9) nothing essential happens, except these two variable swaps. Auxiliary h, f (on the right) and intermediate 0's (in the bottom left corner) are added for finite termination, as discussed in Section 19.2.

It is crucial that diagram (9) can be completed with the $\swarrow$ rewrite step by using one of the shortcut rules $(\swarrow_{1,2})$ (which do not make any variable swaps!) if and only if simultaneously:

(1) $x = u$ and $y = v$, i.e., iff registers are operated *correctly* in the transition from the ID $\langle x, y, \underline{i}\rangle$ to the ID $\langle s(u), v, z\rangle$ and

(2) (a) either $\underline{i}$ in (9) equals $s0$ (in this case $x = y = u = v = 0$) and the rule $(\swarrow_1)$ works,

(b) or $\underline{i}$ in (9) is greater than one (i.e., equals ssz for some z), but the whole term $t \equiv c(\langle x, y, \underline{i}\rangle, c(\langle s(u), v, z\rangle, w))$ in the upper left corner of (9) occurs *in the tail of some embedding list*, i.e., t occurs in $c(t', t)$ for some t', so that $(\swarrow_2)$ could apply.

Remark 8. This double trick is an example of how the commutation of rewrite diagrams is useful to check the needed properties of terms. The first one shows how to check that registers are operated correctly, and the second one ensures that a list starts with the initial ID $\langle 0, 0, 1\rangle$ (otherwise, the commutation by $(\swarrow_{1,2})$ in the head of the list is impossible).

Remark 9. Note that we add three rules of the form (9) for each command $i : AL$ in the program P.

Remark 10. Note that rules (9) do not attempt to check the right succession of commands in the transitions: the third argument in the second triple is a variable z. Another group of rules, described in Sections 13, 14, and 15, will be responsible for this control flow check.

This remark also applies to the analogous rules (9)–(14) below.

To give a better understanding of the above rules, consider two examples.

12.3.2. *Example of a Correct Register Operation*

If P contains the command $8 : AL$ then in the transition from the ID $\langle 6, 4, 8\rangle$ to the ID $\langle 7, 4, 9\rangle$ the registers are operated correctly, and the following rewrite diagram takes place:

$$\begin{array}{rcl} c(u, c(\langle 6, 4, 8\rangle, c(\langle s(6), 4, s(8)\rangle, w))) & \rightarrow & c(u, c(h(\langle 6, 4, 8\rangle, \langle s(6), 4, s(8)\rangle), w)) \\ \downarrow & \swarrow & \Downarrow \\ c(u, c(\langle 6, 4, 8\rangle, c(0, c(\langle s(6), 4, s(8)\rangle, c(0, w))))) & \leftarrow & c(u, c(f(\langle 6, 4, 8\rangle, \langle s(6), 4, s(8)\rangle), w)) \end{array}$$

Here the $\Downarrow$ rewrite is possible by the auxiliary rule $(\Downarrow)$ and the $\swarrow$ rewrite by the shortcut rule $(\swarrow_2)$.

12.3.3. *Example of an Incorrect Register Operation*

If P contains the command $11 : AL$ then in the transition from the ID $\langle 6, 4, 11\rangle$ to $\langle 9, 4, 12\rangle$ the left register is operated incorrectly, and the following rewrite diagram

$$\begin{array}{ccc} c(u, c(\langle 6,4,11\rangle, c(\langle s(8),4,s(11)\rangle, w))) & \to & c(u, c(h(\langle 8,4,11\rangle, \langle s(6),4,s(11)\rangle), w)) \\ \downarrow & \swarrow & \Downarrow \\ c(u, c(\langle 6,4,11\rangle, c(0, c(\langle s(8),4,s(11)\rangle, c(0,w))))) & \leftarrow & c(u, c(f(\langle 8,4,11\rangle, \langle s(6),4,s(11)\rangle), w)) \end{array}$$

cannot be commuted any more by the diagonal $\swarrow$ rewrite using $(\swarrow_2)$, nor by any other rewrite rule.

12.3.4. *Right Addition Command*

The command $i : AR$ is translated into the rules analogous to (9), with $s(\,)$ shifted from the first to the second argument in the second $\langle \ldots \rangle$ of each rule side, namely:

$$\begin{array}{ccc} c(\langle x,y,i\rangle, c(\langle u, s(v), z\rangle, w)) & \to & c(h(\langle u,v,i\rangle, \langle x, s(y), z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x,y,i\rangle, c(0, c(\langle u, s(v), z\rangle, c(0,w)))) & \leftarrow & c(f(\langle u,v,i\rangle, \langle x, s(y), z\rangle), w) \end{array} \tag{10}$$

The intuition behind these rules is clear from the definition of the 2RM and is similar to the rules for the left addition.

12.4. Subtraction Commands

12.4.1. *Left Subtraction*

Quite similarly, a command $i : SL, j$ is translated into two groups of rules, the first three corresponding to the nonzero left register

$$\begin{array}{ccc} c(\langle s(x),y,i\rangle, c(\langle u,v,z\rangle, w)) & \to & c(h(\langle s(u),v,i\rangle, \langle x,y,z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle s(x),y,i\rangle, c(0, c(\langle u,v,z\rangle, c(0,w)))) & \leftarrow & c(f(\langle s(u),v,i\rangle, \langle x,y,z\rangle), w) \end{array} \tag{11}$$

and the second three corresponding to the *empty* left register

$$\begin{array}{ccc} c(\langle 0,y,i\rangle, c(\langle u,v,z\rangle, w)) & \to & c(h(\langle u,v,i\rangle, \langle 0,y,z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle 0,y,i\rangle, c(0, c(\langle u,v,z\rangle, c(0,w)))) & \leftarrow & c(f(\langle u,v,i\rangle, \langle 0,y,z\rangle), w) \end{array} \tag{12}$$

12.4.2. *Right Subtraction*

An instruction $i : SR, j$ is translated analogously into six rules:

$$\begin{array}{ccc} c(\langle x,s(y),i\rangle, c(\langle u,v,z\rangle, w)) & \to & c(h(\langle u,s(v),i\rangle, \langle x,y,z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x,s(y),i\rangle, c(0, c(\langle u,v,z\rangle, c(0,w)))) & \leftarrow & c(f(\langle u,s(v),i\rangle, \langle x,y,z\rangle), w) \end{array} \tag{13}$$

$$\begin{array}{ccc} c(\langle x,0,i\rangle, c(\langle u,v,z\rangle, w)) & \to & c(h(\langle u,v,i\rangle, \langle x,0,z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x,0,i\rangle, c(0, c(\langle u,v,z\rangle, c(0,w)))) & \leftarrow & c(f(\langle u,v,i\rangle, \langle x,0,z\rangle), w) \end{array} \tag{14}$$

The intuition behind these rules is clear from the definition of the 2RM, as for the addition commands.

At this point the reader is invited to stop and get convinced that the rules introduced work exactly in a way required by diagrams (5), (6) in Section 10.

12.5. Checking Correctness of Register Manipulation

The intention behind the rules we constructed so far is better clarified by the following claim (we call it a claim, because it depends on an incompletely defined rewrite rule system). It shows how rewrite diagrams created by rules (9), (10), (11), (12), (13), (14), and commuted by ($\Downarrow$), ($\swarrow$), are used to check whether 2RM's registers are operated correctly along a quasi-correct run (formally explained in the next sections).

Claim 11. [Adequacy]

(1) Let r be a correct run (8) of the 2RM on a program P starting with ID $\langle 0, 0, 1\rangle$. Then the following formula is true (where the predicate R is interpreted as a one step ground reducibility relation in the rewrite rule system $\mathrm{R} = \mathrm{R}(P)$ we are constructing):

$$E(r) \equiv_{df} \forall w_0, w_1, w_2(R(r, w_0) \wedge R(r, w_2) \wedge R(w_2, w_1) \wedge R(w_1, w_0) \Rightarrow R(w_2, w_0)). \qquad (7)$$

(2) Let r be a sequence (8) in which x_i's, y_i's, z_i's are natural numbers, the control flow in r be correct (see below Sections 13 and 14), and $E(r)$ be true. Then r represents a correct run of the 2RM on P starting with ID $\langle 0, 0, 1\rangle$.

The validity of this claim, useful as a guideline for the further development, will be guaranteed by the construction of the remaining part of the rewrite system. We return back to the formal proof of this claim in Section 19. The reader is invited to check that the first part of the claim is true for the part of the system we constructed so far.

Note that the formula (7) is uniform, it does not depend on a program P.

13. QUASI-CORRECT RUNS

We are looking for ground terms r witnessing the truth of the sentence (4) among terms of a special structure, representing right-flattened lists of triples of natural numbers of the form (8). The construction of the formula $E(r)$ in (7) assumes that a term r is quasi-correct. Otherwise $E(r)$ may be true for senseless terms such as $c(c(a, b), c(\varepsilon, h(a, b, d)))$. This is because the rewrite rules we defined so far *do not apply* to such terms; hence, the premise of (7) is false. It is the role of the subformula $C_1(r) \wedge C_2(r) \wedge C_3(r)$ of (4) to detect such senseless cases and become false, so as not to admit false witnesses for (4) satisfying $E(r)$.

The next definition partially captures the idea of correctness.

DEFINITION 12. Call a term r *quasi-correct* if and only if it satisfies the following groups of constraints.

Structural constraints. The term r does not contain subterms of the form:

(1) $h(u, v)$, $f(u, v)$,
(2) $s(F(\dots))$ with $F \in \Sigma\backslash\{s, 0\}$,
(3) $\langle F(\dots), u, v\rangle$, $\langle u, F(\dots), v\rangle$, $\langle u, v, F(\dots)\rangle$ with $F \in \Sigma\backslash\{0, s\}$,
(4) $c(F(\dots), x)$ with $F \in \Sigma\backslash\{\langle, ,\rangle\}$,
(5) $c(x, F(\dots))$ with $F \in \Sigma\backslash\{c, \varepsilon\}$.

(Reason: by definition, a run should be a right-flattened list of triples of natural numbers; thus all subterms enumerated above make no sense in a valid run.)

Boundary constraints. The term r does not contain subterms of the form:

(1) $c(\langle x, y, \underline{j}\rangle, \varepsilon)$ for $1 \leq j < p$

(Reason: a run should end with $c(\langle x, y, \underline{p}\rangle, \varepsilon)$, i.e., after executing p : Halt, the last command in P);

(2) $c(\langle x, y, s^p(z)\rangle, c(\langle u, v, w\rangle, w'))$

(Reason: in a correct run command numbers do not exceed p, command labeled p may (and by the previous constraint should) occur only in the end of the run, i.e., in a subterm $c(\langle x, y, s^p(0)\rangle, \varepsilon)$);

(3) $c(\langle x, y, z\rangle, c(\langle u, v, 1\rangle, w))$

(Reason: in a correct run the control never returns back to the first command; thus label 1 may occur at most once in the beginning; recall Remark 6);

(4) $\langle x, y, 0\rangle$

(Reason: command numbers are positive.)

Control flow constraints. The term r does not contain adjacent triples[9]:

(1) $\langle x, y, i\rangle, \langle u, v, j\rangle$ with $j \neq i+1$ when P contains a command $i : AL$ or $i : AR$.

(Reason: addition transfers control to the next command.)

(2) $\langle x, y, i\rangle, \langle 0, v, z\rangle$ when P contains $i : AL$.

(3) $\langle x, y, i\rangle, \langle u, 0, z\rangle$ when P contains $i : AR$.

(Reason: addition cannot result with the empty register.)

(4) $\langle s(x), y, i\rangle, \langle u, v, j\rangle$ with $j \neq i+1$ when P contains the command $i : SL, i+1$.

(5) $\langle x, s(y), i\rangle, \langle u, v, j\rangle$ with $j \neq i+1$ when P contains the command $i : SR, i+1$.

(Reason: such subtractions, with nonzero registers, always transfer control to the next command.)

(6) $\langle s(x), y, i\rangle, \langle u, v, j\rangle$ with $j = i+1$ when P contains instruction $i : SL, l$ with $l \neq i+1$.

(7) $\langle x, s(y), i\rangle, \langle u, v, j\rangle$ with $j = i+1$ when P contains instruction $i : SR, l$ with $l \neq i+1$.

(Reason: when the left (right) register is positive, such subtractions transfer control to the specified command $l \neq i+1$.)

(8) $\langle 0, y, i\rangle, \langle u, v, j\rangle$ with $j \neq i+1$ when P contains instruction $i : SL, l$ with $l \neq i+1$.

(9) $\langle x, 0, i\rangle, \langle u, v, j\rangle$ with $j \neq i+1$ when P contains instruction $i : SR, l$ with $l \neq i+1$.

(Reason: when the left (right) register is zero, such subtractions transfer control to the succeeding command.)

Remark 13. The Definition 12 of quasi-correctness does not exclude some degenerate cases. Namely, a quasi-correct run r may have one of the forms (and these are all possible cases) enumerated below:

(1) a, b, d,

(2) ε,

(3) 0,

(4) $r \equiv s(r')$ for some r' built of 0 and s,

(5) $r \equiv \langle r_1, r_2, r_3\rangle$ for some r_1, r_2, r_3 built of 0 and s,

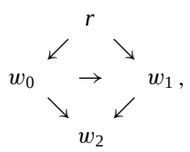
(6) r may be a right-flattened list of triples of natural numbers, with a correct flow control (as defined by the control flow constraints), ending correctly, but possibly with incorrect register manipulations.

Intermediate goal. In the following sections we first show how to determine whether a term is quasi-correct and then proceed to excluding all (degenerate) cases, except the last one.

14. DETERMINING QUASI-CORRECT RUNS

We are going to introduce new rewrite rules that would allow us to reduce every non-quasi-correct term r (see Definition 12) in the following specific way

[9] Say that in the list representation (8) of a run the triples $\langle x, y, i\rangle, \langle u, v, j\rangle$ are adjacent iff they occur in a subterm $c(\langle x, y, i\rangle, c(\langle u, v, j\rangle, w))$.



(15)

which will be impossible for a quasi-correct term.

Consequently, quasi-correct terms r will satisfy the following formula

$$C_1(r) \equiv_{df} \neg\exists\, w_0, w_1, w_2(R(r, w_0) \wedge R(r, w_1) \wedge R(w_0, w_1) \wedge R(w_0, w_2) \wedge R(w_1, w_2)). \tag{16}$$

Remark 14. It is important to note that $C_1(r)$ is equivalent to a universal formula with the quantifier prefix $\forall\forall\forall$, which is essential for keeping the entire sentence H in (4) in the $\exists\forall\forall\forall$-form. We keep the $\neg\exists\exists\exists$-form in (16) as being more intuitive.

Remark 15. The terms $a, b, d, \varepsilon, 0, s^k(0), \langle s^k(0), s^l(0), s^m(0)\rangle$ enumerated as degenerate cases 1–5 in Remark 13 also satisfy both $C_1(r)$ in (16) and $E(r)$ in (7). We exclude these terms by formulas $C_{2,3}(r)$ in Section 16.

15. REWRITE RULES TO CHECK QUASI-CORRECTNESS

The key idea is to define, for each ground term t that cannot be a subterm of a quasi-correct term, two rules: $t \to a$, $t \to b$, plus three (common) rules

$$\begin{array}{l} a \to b, \\ a \to d, \\ b \to d. \end{array} \tag{17}$$

Thus, every term r that is not quasi-correct will form the above diamond-like rewrite diagram (15) and will satisfy the formula $C_1(r)$ defined by (16). Additional effort is needed to ensure that correct terms cannot form the above diamond diagram and thus cannot satisfy $C_1(r)$. Thus the diamond diagram property (16) and the corresponding formula $C_1(r)$ given by (16) will be used as a quasi-correctness criterion.

15.1. Rules for Structural Constraints

By $t \to a, b$ we abbreviate two rules $t \to a$ and $t \to b$. We enumerate the rules for checking structural constraints, corresponding to cases of Definition 12.

(1) A quasi-correct run cannot contain functional symbols $h,\ f$, thus:

$$h(x, y) \to a, b \tag{18}$$

$$f(x, y) \to a, b. \tag{19}$$

(2) $s(F(\ldots)) \to a, b$ for all $F \in \Sigma\backslash\{s, 0\}$.

- (Reason: terms constructed with 0, s are natural numbers and cannot contain subterms starting with something except 0, s.)

(3) (a) $\langle F(\ldots), u, v\rangle \to a, b$ for all $F \in \Sigma\backslash\{0, s\}$,
(b) $\langle u, F(\ldots), v\rangle \to a, b$ for all $F \in \Sigma\backslash\{0, s\}$,
(c) $\langle u, v, F(\ldots)\rangle \to a, b$ for all $F \in \Sigma\backslash\{0, s\}$.

- (Reason: the only meaningful function symbols in the argument positions to the triple constructor $\langle , , \rangle$ are 0 and s.)

(4) (a) $c(F(\ldots), x) \to a, b$ for every $F \in \Sigma\backslash\{\langle, ,\rangle\}$,
(b) $c(x, F(\ldots)) \to a, b$ for every $F \in \Sigma\backslash\{c, \varepsilon\}$.

- (Reason: runs are right-flattened lists (sequences) of triples.)

15.2. Rules for Boundary Constraints

(1) (a) $c(\langle x, y, \underline{j}\rangle, \varepsilon) \to a, b$ for all $1 \le j < p$;
(b) $c(\langle x, y, s^p(z)\rangle, c(\langle u, v, w\rangle, w')) \to a, b$.

- (Reason: the only command that may and should terminate a correct run is $p : Halt$ and thus label p cannot appear in the middle of a run; labels of commands do not exceed p.)

(Note: these two rewrite rules force every right-flattened list of triples of natural numbers to terminate with $c(\langle u, v, s^p(0)\rangle, \varepsilon)$, i.e., with *Halt*, as needed.)

(2) $c(\langle x, y, z\rangle, c(\langle u, v, 1\rangle, w)) \to a, b$

- (Reason: a command with label 1 is executed only in the beginning of a run and the control never returns back to this command; the shortcut with ($\swarrow_1$) will guarantee that the initial ID of a run is $\langle 0, 0, 1\rangle$; see Sections 12.3.1 and 19.)

(3) $\langle x, y, 0\rangle \to a, b$

- (Reason: command numbers are positive.)

15.3. Rules for Control Flow Constraints

Here we again use Convention 2 on mixing sequential and list notation:

(1) (a) $\langle x, y, \underline{i}\rangle, \langle u, v, \underline{j}\rangle \to a, b$ for all j satisfying $1 \le j \ne i + 1 \le p$, when $i : AL$ or $i : AR$ is in P.

- (Reason: addition transfers control to the next command.)

(b) $\langle x, y, \underline{i}\rangle, \langle 0, v, z\rangle \to a, b$ when $i : AL$ occurs in P.
(c) $\langle x, y, \underline{i}\rangle, \langle u, 0, z\rangle \to a, b$ when $i : AR$ occurs in P.

- (Reason: addition cannot result with the empty register.)

(2) (a) If P contains $i : SL$, $i + 1$ add the rules

$$\langle x, y, \underline{i}\rangle, \langle u, v, \underline{k}\rangle \to a, b$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$.

- (Reason: such subtractions *always* transfer control to the succeeding command.)

(b) If P contains $i : SL$, j for $j \ne i + 1$, add the rules

$$\langle 0, x, \underline{i}\rangle, \langle y, z, k\rangle \to a, b$$
$$\langle s(x), y, \underline{i}\rangle, \langle u, v, l\rangle \to a, b$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$, all $l \in \{1, \ldots, p\}\backslash\{j\}$.

- (Reason: such subtractions can only transfer control to the next command, when the register is zero, or to the jth command, when the register is positive.)

(3) (a) If P contains $i : SR$, $i + 1$ add the rules

$$\langle x, y, \underline{i}\rangle, \langle u, v, \underline{k}\rangle \to a, b$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$.

- (Reason: such subtractions *always* transfer control to the succeeding command.)

(b) If P contains $i : SR,\ j$ for $j \neq i+1$, add the rules

$$\begin{aligned} \langle x, 0, i\rangle, \langle y, z, k\rangle &\rightarrow a, b \\ \langle x, s(y), i\rangle, \langle u, v, l\rangle &\rightarrow a, b \end{aligned}$$

for all $k \in \{1, \ldots, p\}\backslash\{i+1\}$, all $l \in \{1, \ldots, p\}\backslash\{j\}$.

• (Reason: such subtractions can only transfer control to the next command, when the register is zero, or to the jth command, when the register is positive.)

16. EXCLUDING DEGENERATE CASES

We should exclude terms

$$a, b, d, \varepsilon, 0, s^k(0), \langle s^k(0), s^l(0), s^m(0)\rangle$$

enumerated as degenerate cases 1–5 in Remark 13; see also Remark 15.

Recall that these terms satisfy both formulas $C_1(r)$ in (16) and $E(r)$ in (7), but they do not witness correct successful terminating runs of the 2RM. We proceed to excluding them by giving the $\forall\forall\forall$-formula $C_2(r) \wedge C_3(r)$ false for these terms but true for terms representing correct terminating runs of the 2RM.

16.1. Excluding a, b, d

It is easy to exclude the terms a, b, d, because none of them satisfies the formula

$$C_2(r) \equiv_{df} \forall w_0 \neg R(w_0, r), \tag{20}$$

whereas each correct terminating run of the 2RM, if any, satisfies (20), by construction of the rewrite system R. Indeed, a, b, d appear as right-hand sides in the rules of the previous section. At the same time, all the rules we constructed have right-hand sides that cannot occur in a correct run.

The difficulty with the remaining terms $\varepsilon, 0, s^k(0), \langle s^k(0), s^l(0), s^m(0)\rangle$ is as follows. Although they do not represent correct terminating runs, they still satisfy the formula (20).

16.2. Excluding $\varepsilon, 0, s^k(0), \langle s^k(0), s^l(0), s^m(0)\rangle$

Let us introduce additional rewrite rules:

$$\begin{aligned} \varepsilon &\rightarrow d, \\ 0 &\rightarrow d, \\ s(x) &\rightarrow d, \\ \langle x, y, z\rangle &\rightarrow d \end{aligned} \tag{21}$$

and consider the following $\forall\forall\forall$-formula

$$\begin{aligned} C_3(r) \equiv_{df} \forall w_0, w_1, w_2 (&R(w_2, w_1) \wedge R(w_1, w_0) \wedge R(w_2, w_0) \\ &\Rightarrow [R(r, w_0) \Rightarrow R(r, w_2) \vee R(r, w_1)]), \end{aligned} \tag{22}$$

which may be better understood in the diagram notation

$$\forall w_0, w_1, w_2 \left(\begin{array}{ccccccccc} r & & w_2 & & r & \rightarrow & w_2 & & r \\ \downarrow & \swarrow & \downarrow & \Rightarrow & & & & \vee & & \searrow \\ w_0 & \leftarrow & w_1 & & & & & & & & w_1 \end{array} \right).$$

This formula is *false* for all terms ε, 0, $s^k(0)$, and $\langle s^k(0), s^l(0), s^m(0)\rangle$. Indeed, take d, b, a for w_0, w_1, w_2, respectively. By rules (22), every r equal to one of ε, 0, $s^k(0)$, $\langle s^k(0), s^l(0), s^m(0)\rangle$ reduces to $w_0 \equiv d$. Thus all the premises in (22) are true, but none of the terms ε, 0, $s^k(0)$, $\langle s^k(0), s^l(0), s^m(0)\rangle$ reduce to $w_2 \equiv a$ nor to $w_1 \equiv b$. Thus the conclusion of (22) is false and none of ε, 0, $s^k(0)$, $\langle s^k(0), s^l(0), s^m(0)\rangle$ satisfy the formula (22). Consequently, $C_3(r)$ excludes these terms, as needed.

At the same time, any correct run does satisfy the formula (22). In fact, let r be a correct run and w_2, w_1, w_0 be such that

$$\begin{array}{ccc} r & & w_2 \\ \downarrow & \swarrow & \downarrow \\ w_0 & \leftarrow & w_1 \end{array} \tag{23}$$

(i.e., all the premises of (22) are satisfied).

Since r is correct, the only way to obtain w_0 as a result of one step rewriting from r is to apply the rule $\downarrow$ from one of the groups (9)–(14). In fact, an alternative would be to apply the rule $\rightarrow$ from one of the groups (9)–(14), but in this case it would be impossible to get such a w_0 as a result of two rewrites (via w_1) from any w_2. The straightforward case analysis shows that in the diagram (23):

(1) either w_2 results from r by application of the rule $\rightarrow$ from the same group as used to get w_0 from r; in this case the atom $R(r, w_2)$ in the conclusion of (22) is true;

(2) or w_2 coincides with r; in this case the atom $R(r, w_1)$ in the conclusion of (22) is true.

Thus in both cases the formula (22) is true for a correct run r.

16.3. Excluding a One Element List

There remains one more degenerate case to be excluded. Consider a one-element list

$$r \equiv c(\langle i, j, k\rangle, \varepsilon),$$

where i, j, k are natural numbers. Obviously, such a list does not represent a correct terminating run of the 2RM. Let us see what happens with the sentence H in this case.

If the number k corresponding to the command label is different from p (the number of commands in the program P), then one of the rules (31), (32) applies and the formula $C_1(r)$ becomes false. Thus such a one-element list is correctly excluded.

However, in the case of $k = p$ neither rules (31) or (32) nor any other rules apply to $c(\langle i, j, p\rangle, \varepsilon)$ any more. Consequently, the formula $C_1(r) \wedge C_2(r) \wedge C_3(r)$ is true. Moreover, the formula $E(r)$ is also true, because $r \equiv c(\langle i, j, p\rangle, \varepsilon)$ is irreducible to satisfy the premises of $E(r)$, hence the premises of $E(r)$ are false. Thus the validity of H is witnessed by a senseless term $c(\langle i, j, p\rangle, \varepsilon)$ that does not represent a correct terminating run of the 2RM.

To deal with this problem we make the list $r \equiv c(\langle i, j, p\rangle, \varepsilon)$ reducible similarly to the case of any two adjacent triples of natural numbers. This is achieved by introducing the following group of rules

$$\begin{array}{ccc} c(\langle x, y, \underline{p}\rangle, \varepsilon) & \rightarrow & c(h(\langle x, y, \underline{p}\rangle, \langle 0, 0, 0\rangle), \varepsilon) \\ \downarrow & & \Downarrow \\ c(\langle x, y, \underline{p}\rangle, c(0, c(\langle 0, 0, 0\rangle, c(0, \varepsilon)))) & \leftarrow & c(f(\langle x, y, \underline{p}\rangle, \langle 0, 0, 0\rangle), \varepsilon) \end{array} \tag{24}$$

similar to groups (9)–(14).

Now, the one-element list $r \equiv c(\langle i, j, p\rangle, \varepsilon)$ creates the rewrite diagram

$$\begin{array}{ccc} . & \rightarrow & . \\ \downarrow & & \downarrow \\ . & \leftarrow & . \end{array}$$

satisfying the premises of $E(r)$. But the conclusion of $E(r)$ is not satisfied by r, because the shortcut rule ($\swarrow_2$) does not apply to a one-element list.

Thus the degenerate case of a one-element list is also excluded.

17. ALL IMPORTANT FORMULAS

Here we repeat verbatim the definition of the sentence H (expressing halting of the 2RM; see Section 9) and its subformulas $E(r)$, $C_{1,2,3}(r)$. All of these formulas are *fixed and independent* of a 2RM program P.

$$H \equiv_{df} \exists r(C_1(r) \wedge C_2(r) \wedge C_3(r) \wedge E(r)). \tag{4}$$

$$E(r) \equiv_{df} \forall w_0, w_1, w_2(R(r, w_0) \wedge R(r, w_2) \wedge R(w_2, w_1) \wedge R(w_1, w_0) \Rightarrow R(w_2, w_0)). \tag{7}$$

$$C_1(r) \equiv_{df} \neg\exists w_0, w_1, w_2(R(r, w_0) \wedge R(r, w_1) \wedge R(w_0, w_1) \wedge R(w_0, w_2) \wedge R(w_1, w_2)). \tag{16}$$

$$C_2(r) \equiv_{df} \forall w_0 \neg R(w_0, r). \tag{20}$$

$$C_3(r) \equiv_{df} \forall w_0, w_1, w_2(R(w_2, w_1) \wedge R(w_1, w_0) \wedge R(w_2, w_0) \Rightarrow [R(r, w_0) \Rightarrow R(r, w_2) \vee R(r, w_1)]). \tag{22}$$

Here R is the binary predicate symbol of the language for the one step rewriting relation (see Section 3). Note that this is the only non-logical symbol in the above formulas.

Remark 16. Equation (4) is in the $\exists\forall\forall\forall$-form, after transformation of (16) into an equivalent $\forall\forall\forall$-form and putting all universal quantifiers (which distribute over $\wedge$) in the prefix.

18. ALL REWRITE RULES

Each program P determines its own rewrite rule system R, as contrasted with the fixed sentence H (see the previous section). Here we summarize (repeat verbatim from the previous sections) all the rewrite rules constructed from a given program.

Let P be an arbitrary but fixed program for the 2RM with $p \geq 2$ instruction numbered consecutively from 1 to p, with the first command $1 : AL$ and containing no commands $i : SL, 1$ or $i : SR, 1$ (see Remark 6). Note that for a fixed 2RM-program P, for each $i \in \{1, \ldots, p\}$ the command labeled i is completely determined. Thus for every $i = 1, \ldots, p-1$, we define the rewrite rules by case analysis depending on the command type, i.e., left addition, right addition, left subtraction, right subtraction (the first command being $1 : AL$ and the last command $p : Halt$).

Some of the rules below, like ($\Downarrow$), are fixed and do not depend on P. Others, like (10), are added to R iff $i : AR$ occurs in P. The rewrite system R will contain as many groups of rules (9), as the program P contains the left addition commands (one group with fixed $\underline{i}$ per command $i : AL$ with label i). Two groups of rules (11), (12) are added for every i such that P contains $i : SL, j$. (And analogously for right addition/subtraction commands.)

Auxiliary rule.

$$\begin{array}{c} h(u, v) \\ \downarrow \\ f(u, v) \end{array} \tag{$\Downarrow$}$$

Rules for the left addition $i : AL$.

$$\begin{array}{rcl} c(\langle x, y, \underline{i}\rangle, c(\langle s(u), v, z\rangle, w)) & \rightarrow & c(h(\langle u, v, \underline{i}\rangle, \langle s(x), y, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x, y, \underline{i}\rangle, c(0, c(\langle s(u), v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, v, \underline{i}\rangle, \langle s(x), y, z\rangle), w) \end{array} \tag{9}$$

Rules for the right addition $i : AR$.

$$\begin{array}{rcl} c(\langle x, y, i\rangle, c(\langle u, s(v), z\rangle, w)) & \to & c(h(\langle u, v, i\rangle, \langle x, s(y), z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x, y, i\rangle, c(0, c(\langle u, s(v), z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, v, i\rangle, \langle x, s(y), z\rangle), w) \end{array} \tag{10}$$

Rules for the left subtraction $i : SL$, j (nonempty register).

$$\begin{array}{rcl} c(\langle s(x), y, i\rangle, c(\langle u, v, z\rangle, w)) & \to & c(h(\langle s(u), v, i\rangle, \langle x, y, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle s(x), y, i\rangle, c(0, c(\langle u, v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle s(u), v, i\rangle, \langle x, y, z\rangle), w) \end{array} \tag{11}$$

Rules for the left subtraction $i : SL$, j (empty register).

$$\begin{array}{rcl} c(\langle 0, y, i\rangle, c(\langle u, v, z\rangle, w)) & \to & c(h(\langle u, v, i\rangle, \langle 0, y, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle 0, y, i\rangle, c(0, c(\langle u, v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, v, i\rangle, \langle 0, y, z\rangle), w) \end{array} \tag{12}$$

Rules for the right subtraction $i : SR$, j (nonempty register).

$$\begin{array}{rcl} c(\langle x, s(y), i\rangle, c(\langle u, v, z\rangle, w)) & \to & c(h(\langle u, s(v), i\rangle, \langle x, y, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x, s(y), i\rangle, c(0, c(\langle u, v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, s(v), i\rangle, \langle x, y, z\rangle), w) \end{array} \tag{13}$$

Rules for the right subtraction $i : SR$, j (empty register).

$$\begin{array}{rcl} c(\langle x, 0, i\rangle, c(\langle u, v, z\rangle, w)) & \to & c(h(\langle u, v, i\rangle, \langle x, 0, z\rangle), w) \\ \downarrow & & \Downarrow \\ c(\langle x, 0, i\rangle, c(0, c(\langle u, v, z\rangle, c(0, w)))) & \leftarrow & c(f(\langle u, v, i\rangle, \langle x, 0, z\rangle), w) \end{array} \tag{14}$$

Shortcut rules (to check whether registers operated correctly).

$$\begin{array}{rcl} & & [h(\langle 0, 0, s0\rangle, \langle 1, 0, v\rangle), \ldots] \\ & \swarrow & \\ [\langle 0, 0, s0\rangle, 0, \langle 1, 0, v\rangle, 0, \ldots] & & \end{array} \tag{$\swarrow_1$}$$

$$\begin{array}{rcl} & & [u, h(\langle x', y', ssz\rangle, \langle x, y, v\rangle), \ldots] \\ & \swarrow & \\ [u, \langle x', y', ssz\rangle, 0, \langle x, y, v\rangle, 0, \ldots] & & \end{array} \tag{$\swarrow_2$}$$

These rules are abbreviations (using list notation) of the following two rules:

$$\begin{array}{rcl} & & c(h(\langle 0, 0, s0\rangle, \langle 1, 0, v\rangle), w) \\ & \swarrow & \\ c(\langle 0, 0, s0\rangle, c(0, c(\langle 1, 0, v\rangle, c(0, w)))) & & \end{array}$$

$$\begin{array}{rcl} & & c(u, c(h(\langle x', y', ssz\rangle, \langle x, y, v\rangle), w)) \\ & \swarrow & \\ c(u, c(\langle x', y', ssz\rangle, c(0, c(\langle x, y, v\rangle, c(0, w))))) & & \end{array}$$

Auxiliary quasi-correctness rules.

$$\begin{array}{l} a \to b, \\ a \to d, \\ b \to d. \end{array} \tag{17}$$

$$h(x, y) \to a, b \tag{18}$$

$$f(x, y) \to a, b \tag{19}$$

Additional rules to exclude ε, $s^k(0)$, $\langle x, y, z\rangle$.

$$\begin{aligned} \varepsilon &\to d, \\ 0 &\to d, \\ s(x) &\to d, \\ \langle x, y, z\rangle &\to d \end{aligned} \tag{21}$$

Additional rules to exclude one element lists.

$$\begin{array}{ccc} c(\langle x, y, \underline{p}\rangle, \varepsilon) & \to & c(h(\langle x, y, \underline{p}\rangle, \langle 0, 0, 0\rangle), \varepsilon) \\ \downarrow & & \Downarrow \\ c(\langle x, y, \underline{p}\rangle, c(0, c(\langle 0, 0, 0\rangle, c(0, \varepsilon)))) & \leftarrow & c(f(\langle x, y, \underline{p}\rangle, \langle 0, 0, 0\rangle), \varepsilon) \end{array} \tag{24}$$

Rules to check structural constraints.

$$s(F(\ldots)) \to a, b \qquad \text{for all } F \in \Sigma\backslash\{s, 0\} \tag{25}$$

$$\langle F(\ldots), u, v\rangle \to a, b \qquad \text{for all } F \in \Sigma\backslash\{0, s\} \tag{26}$$

$$\langle u, F(\ldots), v\rangle \to a, b \qquad \text{for all } F \in \Sigma\backslash\{0, s\} \tag{27}$$

$$\langle u, v, F(\ldots)\rangle \to a, b \qquad \text{for all } F \in \Sigma\backslash\{0, s\} \tag{28}$$

$$c(F(\ldots), x) \to a, b \qquad \text{for every } F \in \Sigma\backslash\{\langle, ,\rangle\} \tag{29}$$

$$c(x, F(\ldots)) \to a, b \qquad \text{for every } F \in \Sigma\backslash\{c, \varepsilon\} \tag{30}$$

Rules to check boundary constraints.

$$c(\langle x, y, \underline{j}\rangle, \varepsilon) \to a, b \qquad \text{for all } 1 \le j < p \tag{31}$$

$$c(\langle x, y, s^p(z)\rangle, c(\langle u, v, w\rangle, w')) \to a, b \tag{32}$$

$$c(\langle x, y, z\rangle, c(\langle u, v, 1\rangle, w)) \to a, b \tag{33}$$

$$\langle x, y, 0\rangle \to a, b \tag{34}$$

Rules to check control flow constraints.

(1) (a) $\langle x, y, \underline{i}\rangle, \langle u, v, \underline{j}\rangle \to a, b$ (35)
for all j satisfying $1 \le j \ne i + 1 \le p$, provided that $i : AL$ or $i : AR$ is in P.

(b) $\langle x, y, \underline{i}\rangle, \langle 0, v, z\rangle \to a, b$ (36)
when $i : AL$ occurs in P.

(c) $\langle x, y, \underline{i}\rangle, \langle u, 0, z\rangle \to a, b$ (37)
when $i : AR$ occurs in P.

(2) (a) If P contains $i : SL$, $i + 1$, then add the rules

$$\langle x, y, \underline{i}\rangle\langle u, v, \underline{k}\rangle \to a, b \tag{38}$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$.

(b) If P contains $i : SL$, j for $j \ne i + 1$, then add the rules

$$\langle 0, x, \underline{i}\rangle\langle y, z, \underline{k}\rangle \to a, b \tag{39}$$

$$\langle s(x), y, \underline{i}\rangle\langle u, v, \underline{l}\rangle \to a, b \tag{40}$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$, all $l \in \{1, \ldots, p\}\backslash\{j\}$.

(3) (a) If P contains $i : SR, i + 1$, then add the rules

$$\langle x, y, \underline{i}\rangle\langle u, v, \underline{k}\rangle \ \rightarrow \ a, b \tag{41}$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$.

(b) If P contains $i : SR, j$ for $j \neq i + 1$, then add the rules

$$\langle x, 0, \underline{i}\rangle\langle y, z, \underline{k}\rangle \ \rightarrow \ a, b \tag{42}$$

$$\langle x, s(y), \underline{i}\rangle\langle u, v, \underline{l}\rangle \ \rightarrow \ a, b \tag{43}$$

for all $k \in \{1, \ldots, p\}\backslash\{i + 1\}$, all $l \in \{1, \ldots, p\}\backslash\{j\}$.

We conclude by a simple property of the constructed term rewriting system R, proved by inspection.

PROPOSITION 17. *Let r be a term representing a correct terminating run of the 2RM. Then only rules $\rightarrow$ and $\downarrow$ from the groups* (9)–(14) *may be applied to r.*

19. THE CORRECTNESS THEOREM

THEOREM 18 (Correctness). *For every 2RM-program P and the associated rewrite rule system* $\mathrm{R} \equiv \mathrm{R}(P)$ (*as described in Section* (18)) *the following four claims are true.*

(1) *The system* R *is* (*left- and right-*) *linear.*

(2) *The system* R *is finitely terminating.*

(3) *The system* R *is confluent.*

(4) *The following two statements are equivalent*:

(*a*) *the 2RM terminates, starting to execute P with the ID* $\langle 0, 0, 1\rangle$;

(*b*) *the sentence H given by* (4) *is true in the first-order theory of one step rewriting generated by* R.

Consequently, there is no general algorithm deciding the $\exists\forall\forall$-theory of one step rewriting for every finite linear canonical system. Henceforth, Part A of the main theorem on weak undecidability holds.

The proof of Theorem 18 occupies the rest of Section 19.

19.1. Proof of Linearity

By immediate inspection of the rules presented in Section 18.

19.2. Proof of Finite Termination

For a term t of signature Σ denote by:

(1) $\#(t, \langle\,\rangle\langle\,\rangle)$ the number of different subterm occurrences of t of the form $c(\langle t_1, t_2, t_3\rangle, c(\langle t_4, t_5, t_6\rangle, t_7))$ (two adjacent triples in a list) and of the form $c(\langle t_1, t_2, t_3\rangle, \varepsilon)$ (a triple adjacent to ε) for some terms $t_{1,2,3,4,5,6,7}$;

(2) $\#(t, F)$ the number of occurrences of the symbol $F \in \{h,\ f, a, b, d\}$ in the term t;

(3) $\#(t, \Sigma)$ the number of occurrences in t of the function symbols from $\Sigma\backslash\{a, b, d\}$.

For a term t of signature Σ denote by $\|t\|$ the ordinal

$$\|t\| \ \equiv_{df} \ \omega^{\omega^{\omega^{\#(t,\langle\rangle\langle\rangle)}}} + \omega^{\omega^{\omega^{\#(t,h)}}} + \omega^{\omega^{\#(t,f)}} + \omega^{\#(t,\Sigma)} + 3 \cdot \#(t, a) + 2 \cdot \#(t, b) + \#(t, d).$$

By inspecting the rewrite rules from Section 18 it can be readily seen that $\|t\| > \|t'\|$ whenever a term t reduces to t' by R. Since ordinals are well ordered, the system R is finitely terminating. Now the role of separating zeros in the first argument positions to the c constructor in all rules (9)–(14) and ($\swarrow_{1,2}$) becomes completely clear. They serve to separate adjacent triples and thus reduce the norms in reductions.

Clearly, we could have used a less strong ordering, but the given proof is conceptually very simple, self-contained, and completely satisfactory for our purposes.

19.3. Proof of Confluence

We assume the reader has basic knowledge about Knuth–Bendix critical pairs algorithm [15–17]. For a finite term rewriting system confluence is equivalent to local confluence, and local confluence is always equivalent to joinability of the so-called critical pairs, easily computable from the so-called superpositions of its left-hand sides.

Here we give a simple proof of the confluence of the constructed rewrite rule system R. Note that the system is quite large (its size varies and depends on the input program P), so we need a kind of meta-argument proving that the system is confluent for every input program P.

Happily, the rewrite rules we constructed possess (intentionally) the following remarkable property, easily checkable by inspection:

> *Every superposition t between rules in* R *always produces a critical pair $\langle t_1, t_2\rangle$ such that both t_1 and t_2 both reduce to d.*

Thus the confluence of R follows by the critical pairs test.

One of the anonymous referees suggested the following more direct and simple proof, without the help of the critical pair lemma. The point is the following fact: every term t can be reduced to d if t is neither variable nor $c(x_1, c(x_2, c(x_3, \ldots c(x_{n-1}, x_{n+1})\ldots)$, where x_i is a variable. Seeing the right-hand side of rewrite rules, we can easily show that any term obtained by one or more rewriting steps is neither variable nor $c(x_1, c(x_2, c(x_3, \ldots c(x_{n-1}, x_{n+1})\ldots)$. Thus, every non-trivial divergence can be joined to d. In this way one does not need to study all the critical pairs of R.

19.4. Proof of (4a) ⇒ (4b)

Let the 2RM terminate, starting to execute the program P in the initial ID $\langle 0, 0, 1\rangle$. We must demonstrate that the sentence H given by (4) is true in the first-order theory of one step rewriting induced by the corresponding system $\mathrm{R} \equiv \mathrm{R}(P)$.

Since the 2RM terminates, there exists a correct run r of the form (3) (represented as a right-flattened list (8) using the c list constructor) starting with $\langle 0, 0, 1\rangle$, ending with $\langle m, n, p\rangle$ (for some natural numbers m, n, p, and p equal the number of commands in P), and such that every transition from the ID $\langle x_i, y_i, z_i\rangle$ to the ID $\langle x_{i+1}, y_{i+1}, z_{i+1}\rangle$ in r is correct with respect to the semantics of the 2RM executing P, as described by Definition 4.

We will now show that this r satisfies the matrix $C_1(r) \wedge C_2(r) \wedge C_3(r) \wedge E(r)$ of (4), which will prove the claim.

Truth of $C_1(r)$. Suppose, toward a contradiction, that $C_1(r)$ is false. Then, by Definition (16) of $C_1(r)$, there exist w_0, w_1, w_2 such that $R(r, w_0) \wedge R(r, w_1) \wedge R(w_0, w_1) \wedge R(w_0, w_2) \wedge R(w_1, w_2)$ is true. Since r is a correct run, only rewrite rules $\rightarrow, \downarrow$ from groups (9)–(14), and no other rules, apply to r (see Proposition 7). Moreover,

(1) by construction of R, the only way to satisfy $R(r, w_0) \wedge R(r, w_1) \wedge R(w_0, w_1)$ is that $r \equiv r[t]$, $w_0 \equiv r[t_0/t]$, $w_1 \equiv r[t_1/t]$ for some terms t, t_0, t_1 such that

$$\begin{array}{l} t \rightarrow t_0 \\ \downarrow \\ t_1 \end{array}$$

where the $\rightarrow$ and $\downarrow$ rewrites are applications of the $\rightarrow$ and $\downarrow$ rules of one of the groups (9)–(14) in the outermost position of t, and the rewrite $w_0 \rightarrow w_1$ is done by one of the shortcut rules ($\swarrow_{1,2}$) (either in

a topmost position of t_0 by ($\swarrow_1$), or by application of ($\swarrow_2$) to $c(t', t_0)$). In fact, if r is (quasi-)correct and

$$\begin{array}{ll} r & \to w_0' \\ \downarrow & \\ w_1' & \end{array}$$

one step rewrite in *different* occurrences of r then, by construction of the rewrite system R, there is no way to shortcut

$$\begin{array}{ll} & w_1' \\ \swarrow & \\ w_0' & \end{array}$$

(2) t_0 may be further reduced in one step to a, or to b (by (29)), or to $c(f(\dots), \dots)$ (by $\Downarrow$)), or to $c(h(\dots), \dots)$ by some rule applied in the second argument position of h, or to $c(a, \dots)$ by (18);

(3) t_1 may only be reduced in one step to terms of the form $c(\langle \dots \rangle, \dots)$;

(4) it follows that $w_0 \equiv r[t_0/t]$ and $w_1 \equiv r[t_1/t]$ cannot be rewritten in one step into the same w_2 so as to satisfy $R(w_0, w_2) \wedge R(w_1, w_2)$, a contradiction.

Truth of $C_2(r)$. The truth of $C_2(r)$ defined by (20) follows by construction of the rewrite system R, because a correct run r cannot be obtained as a result of one step rewrite of any term.

Truth of $C_3(r)$. Let us show the truth of $C_3(r)$ defined by (22). Here we repeat the argument from the end of Section 16.2.

Let r be a correct run and w_2, w_1, w_0 be such that

$$\begin{array}{lll} r & & w_2 \\ \downarrow & \swarrow & \downarrow \\ w_0 & \leftarrow & w_1 \end{array} \tag{44}$$

(i.e., all the premises of (22) are satisfied).

Since r is correct, the only way to obtain w_0 as a result of one step rewriting from r is to apply the rule $\downarrow$ from one of the groups (9)–(14). In fact, an alternative (see Proposition 17) would be to apply the rule $\to$ from one of the groups (9)–(14), but in this case it would be impossible to get such a w_0 as a result of two rewrites (via w_1) from any w_2. The straightforward case analysis shows that in the diagram (44):

(1) either w_2 results from r by application of the rule $\to$ from the same group as used to get w_0 from r; in this case the atom $R(r, w_2)$ in the conclusion of (22) is true;

(2) or w_2 coincides with r; in this case the atom $R(r, w_1)$ in the conclusion of (22) is true.

Thus, in both cases the formula (22) is true for a correct run r.

Truth of $E(r)$. Assume, toward a contradiction, that for a correct run r the formula $E(r)$ defined by (7) is false. Then for some $w_{0,1,2}$ the formula $R(r, w_0) \wedge R(r, w_2) \wedge R(w_2, w_1) \wedge R(w_1, w_0) \wedge \neg R(w_2, w_0)$ is true. Since r is a correct run, only rewrite rules $\to$, $\downarrow$ from groups (9)–(14), or (24) (see Proposition 17), and no other rules apply to r. Moreover,

(1) by construction of the rewrite system R, the only way to satisfy $R(r, w_0) \wedge R(r, w_2) \wedge R(w_2, w_1) \wedge R(w_1, w_0)$ is that for some terms t, t_0, t_1, t_2 one has $r \equiv r[t]$, $w_0 \equiv r[t_0/t]$, $w_1 \equiv r[t_1/t]$, $w_2 \equiv r[t_2/t]$, and

$$\begin{array}{lll} t & \to & t_2 \\ \downarrow & & \Downarrow \\ t_0 & \leftarrow & t_1 \end{array}$$

where all the rewrites, except $\Downarrow$, are done at the topmost position by the rules of one of the groups (9)–(14) or (24);

(2) since r is a correct run, w_2 rewrites to w_0 by one of the shortcut rules ($\swarrow_{1,2}$), i.e., $R(w_2, w_0)$ is necessarily true, and we get a contradiction with the assumption $\neg R(w_2, w_0)$.

19.5. Proof of (4b) $\Rightarrow$ (4a)

Let the sentence H defined by (4) be true in the first-order theory of one step rewriting induced by the rewrite rule system $\mathrm{R} \equiv \mathrm{R}(P)$. We must show that in this case the 2RM terminates, starting to execute P with the ID $\langle 0, 0, 1\rangle$, i.e., that there exists a finite correct run of the 2RM executing P.

Assume r is a term satisfying the matrix $C_1(r) \wedge C_2(r) \wedge C_3(r) \wedge E(r)$ of H. We claim that this r represents a correct terminating run of the 2RM executing P starting from the initial ID $\langle 0, 0, 1\rangle$. In fact, the truth of $C_1(r)$ guarantees that r does not contain subterms matching left-hand sides of rules (18)–(19), (25)–(43) (for structural, boundary, control flow constraints).

(1) Therefore, the term r (cf., Remark 13):

(a) either is one of a, b, d,

(b) or is the empty list ε,

(c) or belongs to the set of natural numbers constructed from 0, s,

(d) or belongs to the set of triples of natural numbers,

(e) or belongs to the set of nonempty right-flattened lists of triples of natural numbers.

(2) The validity of the formula $C_2(r)$ excludes the case (1a); see Section 16.1.

(3) The validity of the formula $C_3(r)$ excludes the cases (1b)–(1d); see Section 16.2.

(4) In the remaining case 1(e) r should be a right-flattened list of triples of natural numbers ending with $\langle i, j, \underline{p}\rangle$ and of length at least 2. In fact, every list satisfying $C_1(r)$ should end with $\langle i, j, \underline{p}\rangle$ (recall rules (31), (32)). By rules (24), such a list creates the rewrite diagram

$$\begin{array}{ccc} . & \rightarrow & . \\ \downarrow & & \downarrow \\ . & \leftarrow & . \end{array}$$

But this diagram can be commuted by the diagonal rewrite $\swarrow$ (to satisfy $E(r)$) using the rule ($\swarrow_2$) only if the list has length ≥ 2. This was our intention with introducing rules (24); see Section 16.3.

(5) By construction of the system R, all subterms of r of the form $c(\langle \ldots\rangle, c(\langle \ldots\rangle, \ldots))$ (i.e., adjacent triples) reduce to form the diagram

$$\begin{array}{ccc} . & \rightarrow & . \\ \downarrow & & \downarrow \\ . & \leftarrow & . \end{array}$$

which commutes by $\swarrow$ since $E(r)$ is true. This commutation guarantees (as we explained in Sections 12.3, 12.5) that all ID transitions in the quasi-correct run r are correct. Recall that the correctness of flow control in r is guaranteed by the validity of $C_1(r)$.

(6) It remains to show that r starts with the initial ID $\langle 0, 0, 1\rangle$. In fact, in the head reduction for the first two triples in the list r we have the rewrite diagram

$$\begin{array}{ccc} . & \rightarrow & . \\ \downarrow & & \downarrow \\ . & \leftarrow & . \end{array}$$

Since it commutes by $\swarrow$ (in the head position), it should necessarily start with the triple $\langle 0, 0, 1\rangle$, because only the list starting with $c(\langle 0, 0, 1\rangle, w)$ can be reduced that way; see rules ($\swarrow_{1,2}$) in Section 12.2 and the related discussion.

(7) Therefore, r is a correct finite successfully terminating run of the 2RM starting with the initial ID $\langle 0, 0, 1\rangle$. This finishes the proof of Theorem 18 and the proof of Part A of our main theorem (weak undecidability).

20. RIGHT-GROUND SYSTEMS

In this section we trade linearity for right-groundedness by briefly sketching how the preceding proof applies (with minor modifications) to show undecidability of the $\exists\forall^3$-theory of one step rewriting in (non-linear) terminating right-ground systems. This was first proved by [11]. Our result is an improvement because of a simpler quantifier prefix ($\exists\forall^3$, as compared with $\exists^2\forall^5$) and a more restricted class of rewrite systems (canonical).

The main idea is as before. We introduce rules corresponding to all commands in the program. Consider a structurally correct run candidate, as before. Assume that the 2RM program in question contains command $i : AL$. To check whether a transition between two adjacent IDs is correctly done by $i : AL$, we have two rules (note that (45) is no longer linear).

$$c(\langle x, y, i\rangle, c(\langle s(x), y, z\rangle, w)) \to A, \tag{45}$$

$$c(\langle x, y, i\rangle, c(\langle u, v, z\rangle, w)) \to B. \tag{46}$$

Similar rules should be added for the right addition and the left and right subtraction; A and B are two new constants not to be confused with the previous ones. We also add the rule

$$B \to A. \tag{47}$$

Consider what happens if a run candidate r contains a correct ID transition using $i : AL$; i.e., $r \equiv r[c(\langle x, y, i\rangle, c(\langle s(x), y, z\rangle, w))]$. Then r reduces both to $r[A]$ and to $r[B]$ by (45), (46), and $r[A]$ reduces to $r[B]$ by (47).

Meanwhile, an incorrect transition in $r \equiv r[c(\langle x, y, i\rangle, c(\langle x', y', z\rangle, w))]$ can be reduced only to $r[B]$ by (46) and not to $r[A]$ (note how non-linearity is useful to check correctness).

Therefore, to check whether a quasi-correct run is correct, write the following formula:

$$E_{rg}(r) \equiv \forall u, v(R(r, u) \wedge R(u, v) \Rightarrow R(r, v)). \tag{48}$$

This should be understood as follows. Suppose, a transition by command i is reducible in r by (46) (it is always reducible this way!) to satisfy $R(r, u)$. Then u is reducible by (47) to satisfy $R(u, v)$. Clearly, if this may be done in one step then the transition reduced in the first step was correct. We leave the straightforward analysis of the other possibilities to the reader.

To achieve confluence (to eliminate critical pairs) we add extra rules like $c(\langle x, y, u\rangle, A) \to B$ and $c(\langle x, y, u\rangle, B) \to B$.

21. STRONG UNDECIDABILITY: FIXED SYSTEMS WITH UNDECIDABLE $\exists\forall^*$-THEORIES

We thus proved the weak undecidability (Part A) of our main theorem (cf. Sections 1, 5) for the $\exists\forall\forall$-theories of one step rewriting. Thus, no algorithm is able to decide the $\exists\forall\forall$-theory of an *arbitrary* finite canonical linear system. On the other hand, whenever *any finite rewriting system* is *fixed*, its $\exists\forall\forall$-theory, $\exists\exists\forall\forall\forall\forall$, etc. (for all quantifier prefixes expressed by regular expressions defining finite languages; see Proposition 3) are decidable.

In this section we present a construction of the fixed canonical linear system with undecidable $\exists\forall^*$-theory of one step rewriting. This is currently the simplest quantifier prefix class for which the strong undecidability of the theories of one step rewriting is known.

The development of this section reuses the machinery developed in the preceding sections and is therefore more schematic, with some trivial and repeating parts left out. As a technical tool we use a

reduction from a slightly different undecidable problem due to [12–14], for the *two-register machines with input*.

THEOREM 19. (Version with input [14, p. 59]). *There exist concrete examples of the "universal" program P such that given a natural number n it is undecidable* (*more precisely*, *r.e.-complete*) *whether or not the* 2*RM halts when started with the first instruction of P and both registers containing the number n.*

Remark 20. The problem remains undecidable when in the statement of Theorem 19 the phrase *a natural number n* is replaced with *a natural number $n > N$* (*where N is any a priori fixed natural number*).

Technically, we need to say that a run candidate starts with an ID $\langle n, n, 1\rangle$ (for any natural $n > N$, where N is some fixed bound), instead of saying that it starts with $\langle 0, 0, 1\rangle$, as we did before. Thus, for every $n > N$ we must construct a formula $S_n(r)$ saying that $r \equiv c(\langle n, n, 1\rangle, w)$ for some w.

The overall sentence expressing halting of the universal 2RM-program P on the number n will have the form

$$H_n \equiv_{df} \exists r(C_1(r) \wedge C_2'(r) \wedge C_3'(r) \wedge E(r) \wedge S_n(r)), \tag{49}$$

where $S_n(r)$ and slightly modified formulas $C_2'(r)$, $C_3'(r)$ are described below.

Note again that unlike the previously *fixed* sentence (4), now the sentences H_n are not going to be fixed any more, and the set of all quantifier prefixes of sentences H_n is going to be *infinite* (recall that this is necessary by Proposition 3). Moreover, each such prefix will belong to $\exists\forall^*$.

21.1. Changes to the Rewrite System

Given a universal 2RM-program P (as guaranteed by Theorem 19; we may still assume that P starts with $1 : AL;\ 2 : SL, 3$) we construct the corresponding rewrite system as before, with the following modification.

Instead of the rule $(\swarrow_1)$ we introduce the modified shortcut rule

$$[\langle x', y', s0\rangle, 0, \langle x, y, v\rangle, 0, \dots] \swarrow [h(\langle x', y', s0\rangle, \langle x, y, v\rangle), \dots] \tag{$\swarrow_1'$}$$

This is needed in order to check correctness of the register manipulation on the first step; recall that the computations now start with $\langle n, n, 1\rangle$ and not with $\langle 0, 0, 1\rangle$ as before.

21.2. Saying that a Run Starts with $\langle n, n, 1\rangle$

Suppose that the existentially quantified in (49) run candidate r is structurally correct, with all correct transitions, correct flow control, and terminating correctly, as before, but we do not insist that it starts with $\langle 0, 0, 1\rangle$.

The general idea to express that it starts with $\langle n, n, 1\rangle$, i.e., has form $r \equiv c(\langle n, n, 1\rangle, w)$, is as follows. We introduce new rewrite rules allowing for the rewrite chains of the form

$$r_n \to \cdots \to r_0 \to r \tag{50}$$

with the property that r has form $c(\langle n, n, 1\rangle, w)$ if and only if $r_n \to r_0$. Note that in contrast with the previous development we now allow a correct run to be obtained as a result of a sequence of rewrite steps. This causes a slight change in the definition of the formulas $C_{2,3}$ below in this section.

First, we augment the rewrite system with the following rules

$$s(c(\langle x, y, s(z)\rangle, w)) \to s(c(\langle s(x), s(y), z\rangle, w)), \tag{51}$$

$$s(c(\langle 0, 0, s(z)\rangle, w)) \to s(c(\langle s(z), s(z), 0\rangle, w)), \tag{52}$$

$$s(c(\langle s(z), s(z), 0\rangle, w)) \to c(\langle s(z), s(z), s(0)\rangle, w), \tag{53}$$

where (53) provides for the last step in the chain (50), (51) allows for the first n steps, and (52) shortcuts $r_n \to r_0$. We add the outermost s in the above rules so as to *localize* possible application of the rules in the *head* of a term.

Take it another way: the rule (51) stepwise pumps the third argument into the first two treating them equally, while (52) does the same in just one step, when started from zeros.

Now for every $n > 0$ and every term $r \equiv c(\langle s^n(0), s^n(0), s(0)\rangle, w)$ we have a *unique* chain (50), where

$$r_i \equiv s(c(\langle s^{n-i}(0), s^{n-i}(0), s^i(0)\rangle, w)), \tag{54}$$

and in this case, indeed, $r_n \to r_0$ in just one step by (52).

We use this property as a *characteristic* one to express *starting with* $\langle n, n, 1\rangle$ by the following formula (where we use $r_i \to r_k$ instead of $R(r_i, r_k)$):

$$S'_n(r) \equiv_{df} \forall r_n, \ldots, r_0(r_n \to \cdots \to r_0 \to r \Rightarrow r_n \to r_0). \tag{55}$$

We are almost done. However, this does not work yet, because when $k < n$ or $j < n$ the term $r = c(\langle s^k(0), s^j(0), s(0)\rangle, w)$ also satisfies (55). This is due to the fact that for n backward rewrite steps from r_0 in (50) one needs at least $k \geq n$ and $j \geq n$. Consequently, the premise of (55) is always false and thus (55) is true for $r = c(\langle s^k(0), s^j(0), s(0)\rangle, w)$ whenever $k < n$ or $j < n$.

Otherwise, the formula (55) is true for $r = c(\langle s^n(0), s^n(0), s(0)\rangle, w)$, because the only possible substitutions for the universally quantified variables to satisfy the premise are given by (54) and $r_n \to r_0$ by (52). Additionally, (55) perfectly excludes all terms $r = c(\langle s^k(0), s^j(0), s(0)\rangle, w)$ with $k, j > n$. This is because for every such term there is exactly one way to satisfy the premise of (55), but in this case the conclusion of (55) fails.

To exclude the terms $r = c(\langle s^k(0), s^j(0), s(0)\rangle, w)$ for $k < n$ or $j < n$, not yet excluded by (55), we introduce the following extra rules. Our intention is to get a fork whenever the backward applications of the rule (51) while creating the chain (50) backwardly gets stuck (one or both arguments become zero) before the n-step chain $r_n \to \cdots \to r_0$ is created.

$$ss(c(\langle 0, s(y), z\rangle, w)) \to s(c(\langle 0, s(y), z\rangle, w)), \tag{56}$$

$$sss(c(\langle 0, s(y), z\rangle, w)) \to s(c(\langle 0, s(y), z\rangle, w)), \tag{57}$$

$$sss(c(\langle 0, s(y), z\rangle, w)) \to ss(c(\langle 0, s(y), z\rangle, w)), \tag{58}$$

and, symmetrically,

$$ss(c(\langle s(x), 0, z\rangle, w)) \to s(c(\langle s(x), 0, z\rangle, w)), \tag{59}$$

$$sss(c(\langle s(x), 0, z\rangle, w)) \to s(c(\langle s(x), 0, z\rangle, w)), \tag{60}$$

$$sss(c(\langle s(x), 0, z\rangle, w)) \to ss(c(\langle s(x), 0, z\rangle, w)), \tag{61}$$

and, to cover the case when both arguments are exhausted simultaneously,

$$ss(c(\langle 0, 0, z\rangle, w)) \to s(c(\langle 0, 0, z\rangle, w)), \tag{62}$$

$$sss(c(\langle 0, 0, z\rangle, w)) \to s(c(\langle 0, 0, z\rangle, w)), \tag{63}$$

$$sss(c(\langle 0, 0, z\rangle, w)) \to ss(c(\langle 0, 0, z\rangle, w)). \tag{64}$$

Therefore, in the case when $r = c(\langle s^k(0), s^j(0), s(0)\rangle, w)$ with $k < n$ or $j < n$, either (56), (57), or (59), (60), or (62), (63) backwardly apply making a fork at a distance $< n$ from r_0. This fork commutes by (58), or (61), or (63), respectively, and the following formula is satisfied for some $l = \min(k, j) < n$:

$$Q_l(r) \equiv_{df} \neg\exists r_l'', r_l', r_l, \ldots, r_0 \left(\begin{array}{ccccccc} & & r_l'' & & & & \\ & \swarrow & & \searrow & & & \\ r & \to \cdots \to & r_l & & \to \cdots \to & r_0 & \to r \\ & \searrow & & \nearrow & & & \\ & & r_l' & & & & \end{array} \right). \tag{65}$$

Note that this formula is equivalent to a universal formula (important for our purposes), but we leave it in a more intuitive form.

Now for every $n > 1$ consider the following formula (also equivalent to a universal formula)

$$S_n''(r) \equiv_{df} \bigwedge_{l=1}^{n-1} Q_l(r), \tag{66}$$

which says that one can create a backward chain (50) of length n without getting forks.

Finally, the needed formula $S_n(r)$ expressing the property that r starts with $\langle n, n, 1\rangle$ may be written as follows

$$S_n(r) \equiv_{df} S_n'(r) \wedge S_n''(r),$$

which is also equivalent to a universal formula, with the number of $\forall$ growing with n.

21.3. Excluding a, b, d

We need to slightly correct the formula $C_2(r)$, see (20), saying that r differs from a, b, d. This is necessary because now, after introduction of the rule (53), a correct run *can be obtained as a result of one step rewrite from another term*. This was not possible before, and we used $C_2(r) \equiv \forall w_0 \neg R(w_0, r)$ to exclude incorrect runs a, b, d; see Section 16.1. If we stay with this $C_2(r)$, it will exclude also the correct runs, after introduction of the new rules in the previous section.

Still, with the new rules the incorrect runs a, b, d are easily excluded, because none of them satisfy the following formula[10]

$$C_2'(r) \equiv_{df} \forall u, u_a, u_b, u_d \left(\begin{array}{ccccc} & & u & & \\ & \swarrow & & \searrow & \\ u_a & & \to & & u_b \\ & \searrow & & \swarrow & \\ & & u_d & & \end{array} \Rightarrow \neg \left[\begin{array}{ccc} & u & \\ \swarrow & & \\ r & \to & u_b \\ \searrow & & \\ & u_d & \end{array} \right] \right.$$
$$\left. \wedge \neg \left[\begin{array}{ccc} & u & \\ & \searrow & \\ u_a & \to & u_b \\ & \swarrow & \\ & r & \end{array} \right] \wedge \neg \left[\begin{array}{ccc} u_a & \to & u_b \\ & \searrow \quad \swarrow & \\ & r & \end{array} \right] \right). \tag{67}$$

Intuitively this formula says whenever u, u_a, u_b, u_d form a diamond diagram as in the premise, which automatically means that u is incorrect and $u_a = a$, $u_b = b$, and $u_d = d$, then r is neither a, nor b, nor c. This is exactly what we need.

[10] We use graphic diagrams here as more intuitive; they can be easily transformed into a strict notation by replacing every diagram in [ ] with a conjunction of atoms $R(x, y)$ corresponding to $x \to y$.

Note that $C_2'(r)$ is one universal quantifier more expensive than $C_2(r)$. Now, as the number of universal quantifiers in the sentences H_n should necessarily (by Proposition 3) grow unboundedly, we can afford being more wasteful than before.

21.4. Excluding ε, $s^k(0)$, $\langle s^k(0), s^l(0), s^m(0)\rangle$

We need to change the formula $C_3(r)$, because the analysis from Section 16.2 (w_0 cannot be obtained from any w_2 by two rewrite steps) does not work any more. Fortunately we can be more wasteful now and use more universal quantifiers (namely, we need four instead of three).

$$C_3'(r) \equiv_{df} \forall u, u_a, u_b, u_d \left(\begin{array}{ccccc} & & u & & \\ & \swarrow & & \searrow & \\ u_a & & \rightarrow & & u_b \\ & \searrow & & \swarrow & \\ & & u_d & & \end{array} \Rightarrow (r \rightarrow u_d \Rightarrow (r \rightarrow u_a \vee r \rightarrow u_b)) \right). \tag{68}$$

When the premise of this formula is satisfied, then necessarily $u_a = a, u_b = b, u_d = d$. Clearly, each of ε, $s^k(0)$, $\langle s^k(0), s^l(0), s^m(0)\rangle$ reduces to d, but none reduces either to a, or to b. Thus, these terms violate $C_3'(r)$. On the other hand, the straightforward analysis shows that all correct runs do satisfy $C_3'(r)$.

This finishes the construction. One can easily check that all the rules we introduced are linear and do not damage the canonicity of the rewrite system. We thus proved Part B of the main theorem on strong undecidability.

22. STRONG UNDECIDABILITY OF THE ∃∀∀∀-THEORIES WHEN FUNCTION SYMBOLS ARE ALLOWED

Recall that by definition of the theories of one step rewriting in Section 3 function symbols were *forbidden* in formulas. This added technical difficulties in expressing quite obvious things (very easy in presence of function symbols) but has not prevented the theories of one step rewriting from being undecidable. In fact, a more natural and liberal definition would have allowed for using function symbols in formulas. In this case the complications we had to deal with in the previous sections disappear, and we obtain the following strong undecidability result for theories of finite quantifier prefix ∃∀∀∀ (without function symbols this is impossible by Proposition 3).

THEOREM 21. *If signature function symbols are allowed in formulas, then there exist finite linear canonical systems with r.e.-complete sets of true prenex sentences of the theory of one step rewriting of the form*

$$\exists r \forall w_1, w_2, w_3\ \Phi(r, w_1, w_2, w_3),$$

where $\Phi(r, w_1, w_2, w_3)$ *is quantifier-free.*

Remark 22. Since the theory of one step rewriting is complete (i.e., every sentence is either true or false), the set of true prenex sentences of the theory of one step rewriting of the form $\forall r \exists w_1, w_2, w_3 \Phi(r, w_1, w_2, w_3)$, where $\Phi(r, w_1, w_2, w_3)$ is quantifier-free, is *co-r.e.-complete*. All the arithmetic hierarchy may now be constructed in the usual manner.

Proof. The sentences H_n defined in (49) may now be defined in the ∃∀∀∀-form

$$H_n \equiv_{df} \exists r \left(\begin{array}{l} E(c(\langle s^n(0), s^n(0), s(0)\rangle, r)) \ \wedge \\ C_1(c(\langle s^n(0), s^n(0), s(0)\rangle, r)) \end{array} \right),$$

where $E(r)$, $C_1(r)$ are ∀∀∀-formulas as before. Note that the additional formulas C_2, C_3 excluding degenerate cases are not needed any more, due to the ability to use functional symbols. ■

23. CONCLUSIONS

In this paper by using reductions from the halting problems for Minsky's two-register machines (inputless and with input) we proved the following undecidability results for the theories of one step rewriting.

(Weak undecidability). There is no general algorithm capable of deciding the $\exists\forall\forall$-theory of one step rewriting for *every* finite linear canonical system (despite the fact that for each such system this theory is decidable non-uniformly).

This improves over previously known results of the same kind due to the use of the simpler quantifier prefix and simultaneously linear and canonical systems.

(Strong undecidability). There exist *fixed* finite linear canonical systems with undecidable (r.e.-complete) $\exists\forall^*$-theories of one step rewriting. If function symbols are allowed in the formulas of the theory, then even the finite prefix class $\exists\forall\forall$ is undecidable. This improves previous results of the author and gives the strongest currently known undecidability result (as per simplicity of the quantifier prefix and restrictedness of the class of rewrite systems).

It remains open whether positive quantified theories of one step rewriting are decidable. Note in this respect that ground reducibility expressed by a positive $\forall^*\exists$-sentence is decidable for the usual rewrite systems [18], but is undecidable for conditional systems, both in the weak sense [19] and in the strong sense [20].[11]

Another problem worth investigating is the non-uniform decidability of theories of one step rewriting with finite prefixes. Given any finite term rewriting system R and a regular expression **Q** over $\{\exists, \forall\}$ describing a finite set of quantifier prefixes, the **Q**-theory of one step rewriting in R is always decidable (Proposition 3). Develop decision algorithms and investigate inherent complexity.

ACKNOWLEDGMENTS

Many thanks to Harald Ganzinger, Uwe Waldmann, and anonymous referees for numerous useful suggestions, ideas, remarks, discussions, and encouragement.

REFERENCES

1. Caron, A.-C., Coquidé, J.-L., and Dauchet, M. (1993), Encompassment properties and automata with constraints, *in* "Rewriting Techniques and Applications'93" (C. Kirchner, Ed.), Lecture Notes in Computer Science, Vol. 690, pp. 328–342, Springer-Verlag, Berlin.
2. Dershowitz, N., Jouannaud, J. P., and Klop, J. W. (1993), More problems in rewriting, *in* "Rewriting Techniques and Applications'93," Lecture Notes in Computer Science, Vol. 690, pp. 468–487, Springer-Verlag, Berlin/New York.
3. Dershowitz, N., Jouannaud, J. P., and Klop, J. W. (1995), Problems in rewriting III, *in* "Rewriting Techniques and Applications'95," Lecture Notes in Computer Science, Vol. 914, pp. 457–471, Springer-Verlag, Berlin/New York.
4. Dauchet, M., Caron, A.-C., and Coquidé, J.-L. (1995), Automata for reduction properties solving, *J. Symbolic Comput.* **20**, 215–233.
5. Dauchet, M., and Tison, S. (1990), The theory of ground rewrite systems is decidable, *in* "Proc 5th IEEE Symp Logic in Computer Science," pp. 242–256.
6. Treinen, R. (1996), The first-order theory of one step rewriting is undecidable, *in* "Rewriting Techniques and Applications'96," Lecture Notes in Computer Science, pp. 276–285, Springer-Verlag, Berlin.
7. Vorobyov, S. (1995), The elementary theory of one-step rewriting is undecidable (note), "On Trees and Rewriting," unpublished draft, *available at* http://www.mpi-sb.mpg/∼sv/.
8. Quine, W. V. (1946), Concatenation as a basis for arithmetic, *J. Symbolic Logic* **11**(4), 105–114.
9. Seynhaeve, F., Tommasi, M., and Treinen, R. (1997), Grid structure and undecidable constraint theories, *in* "TAPSOFT'97," Lecture Notes in Computer Science, Vol. 1214, pp. 357–368, Springer-Verlag, Berlin.
10. Vorobyov, S. (1997), The first-order theory of one step rewriting in linear noetherian systems is undecidable, *in* "Rewriting Techniques and Applications'97," Lecture Notes in Computer Science, Vol. 1232, pp. 254–268, Springer-Verlag, Berlin, *available at* http://www.mpi-sb.mpg.de/∼sv.

Au: please provide volume number of LNCS for [6] and [11]

[11] Kaplan-Choquer showed that there is no algorithm capable of deciding ground confluence in an arbitrary finite decreasing conditional system. Our result is stronger: we construct fixed examples of systems with undecidable ground reducibility. The existence of such systems does not follow from Kaplan-Choquer's result.

11. Marcinkowski, J. (1997), Undecidability of the first-order theory of one step right ground rewriting, *in* "Rewriting Techniques and Applications'97," Lecture Notes in Computer Science, Springer-Verlag, Berlin.
12. Minsky, M. (1961), Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines, *Ann. of Math.* **74**(3), 437–455.
13. Minsky, M. (1967), "Computation: Finite and Infinite Machines," Prentice Hall, New York.
14. Lewis, H. R. (1979), "Unsolvable Classes of Quantificational Formulas," Addison-Wesley, Reading, MA.
15. Huet, G., and Oppen, D. C. (1980), Equations and rewrite rules: A surve, *in* "Formal Language Theory: Perspectives and Open Problems," pp. 349–406, Academic Press, New York.
16. Dershowitz, N., and Jouannaud, J.-P. (1990), Rewrite systems, *in* "Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics" (Jan van Leeuwen, Ed.), pp. 243–320, Elsevier, Amsterdam.
17. Knuth, D., and Bendix, P. (1970), Simple word problems in universal algebras, *in* "Computational Problems in Abstract Algebra" (J. Leech, Ed.), pp. 263–297. Pergamon Press, Oxford.
18. Plaisted, D. (1985), Semantic confluence tests and completion methods, *Inform. and Control* **65**, 182–215.
19. Kaplan, S., and Choquer, M. (1986), On the decidability of ground reducibility, *Bull. EATCS* **28**, 32–34.
20. Vorobyov, S. (1998), A decreasing conditional rewrite system with Π^0_1-complete ground, reducibility, manuscript.