The existential fragment of the one-step parallel rewriting theory

Aleksy Schubert*

Institute of Informatics
Warsaw University
ul. Banacha 2
02–097 Warsaw
Poland

SoS Group
NIII
Faculty of Science
University of Nijmegen
The Netherlands

Abstract. It is known that the first-order theory with a single predicate $\rightarrow$ that denotes one-step rewriting reduction on terms is undecidable already for formulae with $\exists\forall$ prefix. Several decidability results exist for the fragment of the theory in which the formulae start with the $\exists$ prefix only. This paper considers a similar fragment for a predicate $\rightarrow^p$ which denotes the parallel one-step rewriting reduction. We show that the theory is related to the type entailment problem and prove that the first-order theory of $\rightarrow^p$ is undecidable already for formulae with $\exists$ prefix.

1 Introduction

The first-order one-step parallel rewriting theory is a first-order theory which has only one relation symbol $\rightarrow^p$. The logical value of formulae in this theory is checked in a structure of ground terms over a signature Σ. Two terms s, t are in the relation $\rightarrow^p$ when s can be rewritten in one parallel step to t using rewrite rules from a fixed finite set R. It is worth noting that the formulae of the theory cannot use the equality relation $=$ and function symbols from Σ. There is also no direct way to express the fact that a particular rewriting is done with a specific set of rules from R.

The notion of the parallel term rewriting emerged in the studies on computational frameworks [GKM87]. This model of computation allows to investigate computations in the context of concurrent or parallel programming [AK96]. Efficient implementations of parallel rewriting systems rely on various graph rewriting techniques which were intensely studied (for an overview see e.g. [Ass00]). Parallel rewriting has also been used as a basis for the logical framework of rewriting logic [MOM02] as well as in the context of regular tree languages [STT97].

The first-order theory of one-step rewriting, but in the non-parallel case, has been proved to be undecidable by Treinen [Tre96,Tre98]. This result was further strengthened to work for various weak classes of rewriting systems: linear, shallow

* This work was partly supported by KBN grant 3 T11C 002 27 and Sixth Framework Programme MEIF-CT-2005-024306 SOJOURN.

[STT97,STTT01]; linear, terminating [Vor97,Mar97]; right-ground, terminating [Mar97]; and finitely-terminating, linear, confluent [Vor02]. The proofs in (most of) the papers above showed undecidability of $\exists^*\forall^*$ fragment of the theory. The strongest of them was [Tre96] where already $\exists^2\forall$ fragment is undecidable. However, this was not obtained for a fixed rewrite system. Vorobyov in [Vor02] showed a fixed system for which $\exists\forall^3$ fragment is undecidable.

Despite the negative results in the general case, researchers investigated a special case in which the existential formulae of the one-step rewriting theory were considered. The currently existing results show decidability of certain subclasses of the theory. Early papers considered more general results. The theorems there in particular imply that the theory is decidable in case of left-linear right-ground systems [Tis90] and for unary signatures [Jac96]. A more specific consideration of the problem resulted in the decidability for quasi-shallow rewriting systems [CSTT99]; linear, non left-left-overlapping; and non ϵ-left-right-overlapping systems [LR99]. The problem is also decidable for arbitrary rewriting system, but in case the formulae are positive [NPR97].

The parallel rewriting theory we discuss here is also related to the non-structural type entailment problem. This open problem has been extensively studied in the literature [HR97,HR98,NP99,NP03]. We show here that an instance of the type entailment problem can be encoded as a term rewriting system and an existential formula in the one-step parallel rewriting theory.

This paper is organised in the following way. We fix the notation in Section 2. This is followed by a link with the type entailment problem in Section 3. Subsequently, we present the undecidability proof in Section 4. The presentation of the proof is divided into two subsections. The first of them (4.1) presents a slightly modified version of the Turing machine, which is easier to handle in the proof, and the second (4.2) presents a class of rewriting systems that simulates the work of the machine which leads to a proof of undecidability of the rewriting theory we deal with here. We conclude the paper with a discussion in Section 5.

2 Preliminaries

This section recalls preliminary notions used in the rest of the paper and fixes the notation.

The function symbols belong to signatures which are usually denoted by Σ, Σ', etc. Each signature is equipped with an arity function $ar : \Sigma \to \mathbb{N}$. The symbols of non-zero arity in Σ are usually denoted by letters such as f, g etc. The zero arity symbols are denoted by c, d etc. Let X be disjoint with Σ. The symbols from X can be treated as symbols of arity 0 to form terms with variables (open types). The variables are by convention written as x, y etc. When convenient, we do not distinguish between terms, types, and trees labelled with the symbols from Σ or $\Sigma \cup X$. The set of all finite ground terms (closed types) over a signature Σ is denoted by $T(\Sigma)$. The set of terms with variables in X is denoted by $T(\Sigma, X)$. The terms are usually denoted by small Latin letters such as t, s, u, etc. The types are denoted by small Greek letters such as τ, σ, etc. The

set of variables that occur in a term t is denoted by $\mathrm{FV}(t)$. A substitution can replace occurences of variables in a term t with some other terms. Substituations are usually noted as S, T etc. and the result of the application of a substitution S to a term t is written as $S(t)$. We denote by $C[-_1, \ldots, -_n]$ a context which contains $n > 0$ placeholders (each occurring exactly once). The placeholders may be replaced by particular terms $t_1, \ldots, t_n$ which is denoted as $C[t_1, \ldots, t_n]$.

In order to address positions in a tree we use sequences of natural numbers. The addresses are denoted by small Greek letters like γ, ρ, etc. The root address (empty sequence) is denoted by ε. The subtree of t at an address γ is denoted by $t|_\gamma$. We compose addresses so that $f(t_1, t_2)|_{i \cdot \gamma} = t_i|_\gamma$. The result of replacement of the subtree at an address γ in t with a tree s is denoted by $t[\gamma \leftarrow s]$.

Let R be a finite set of pairs of terms $\langle l, r \rangle$ over a signature Σ such that $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. We call such pairs *rewrite rules* over Σ and write them $l \to r$. We say that a term t *rewrites* to a term s with a rule $l \to r$ when there is an address γ in both t and s and a substitution S such that $t = t[\gamma \leftarrow S(l)]$ and $s = t[\gamma \leftarrow S(r)]$. We say that a term t *rewrites* to a term s (written $s \to t$) when t rewrites to s with some rule $l \to r \in R$.

2.1 One-step parallel rewriting theory

This subsection presents the notions concerning the first-order parallel one-step rewriting theory. We fix a signature Σ which contains at least one symbol of arity 0.

Definition 1. `(definition of` $\to^p$`)`
Let R be a set of rewrite rules over Σ. We consider a relational structure $\mathcal{A}_R = \langle T(\Sigma), \to^p \rangle$ where the symbol $\to^p$ represents the one-step parallel rewriting and is defined as follows: $t \to^{p\ \mathcal{A}} s$ for $t, s \in T(\Sigma)$ iff there is a context $C[-_1, \ldots, -_k]$ with $k > 0$ such that $t = C[t_1, \ldots, t_k]$ and $s = C[s_1, \ldots, s_k]$ and for $i = 1, \ldots, k$ we have $t_i \to s_i$. Additionally, we use the symbol $\to^p_*$ to denote the reflexive-transitive closure of $\to^p$.

The atomic formulae of the first-order one-step paralell rewriting theory are of the shape $x \to^p y$ only (no formulae of the form $x = y$). These atomic formulae can be combined with $\neg, \wedge$ and $\vee$. Free variables can be bound by quantifiers $\exists, \forall$. We write $x \not\to^p y$ as a shorthand for $\neg x \to^p y$.

It is worth pointing out that the context C used in the definition above has at least one placeholder. This design solution makes the notion closer to the notion of the usual one-step rewriting as in both cases at least one rewrite rule is executed.

The existential fragment of the first-order one-step parallel rewriting theory consists of closed formulae of the form $\exists x_1 \cdots \exists x_n . \phi$ where ϕ does not contain quantifiers.

This theory is strictly stronger than the non-parallel one i.e. each existential formula ϕ that holds when $\to^p$ is interpreted as the usual one-step rewriting holds also when $\to^p$ is interpreted as the parallel one-step rewriting — one can use the same substitution and the same interpretation of rewrites to witness

the satisfaction of the formula in case of the parallel rewriting. Still, there are formulae which are satisfied with the parallel interpretation, but not satisfied in case of the non-parallel one. Indeed, let $\Sigma = \{f, a, b\}$ where f is binary and a, b are constants. Consider the rewrite system with the rules $f(a,a) \to f(b,b), b \to a, f(x,x) \to f(x,x)$. The formula $\exists x, y.x \to^p x \wedge x \to^p y \wedge y \to^p x \wedge \neg y \to^p y$ holds in case the interpretation with the parallel rewriting is used while it does not hold for the interpretation with the non-parallel one.

The main decision problem we deal with in this paper is the following:

Definition 2. `(satisfiability problem)`
Input: $\langle \Sigma, \phi, R \rangle$ where ϕ is a formula of the first-order one-step rewriting theory and R is a rewrite system over Σ (Σ contains at least one constant).
Question: Is ϕ satisfied in the structure $\mathcal{A}_R$ over the signature Σ?

3 Relation with type entailment

The non-structural type entailment problem can be exploited in type inference engines with subtyping. Such systems operate efficiently when powerful simiplification procedures exist for typings. These simiplification procedures can be expressed in terms of type entailment. In this section we show that the existential fragment of the theory of the one-step parallel rewriting is related to the non-structural type entailment problem in such a way that type entailment instances can be effectively translated to instances of the existential theory. This construction works only in case the binary symbol in the type entailment problem is covariant in both arguments.

Consider type inequalities $\sigma \leq \tau$ between types built of the symbols $\{\bot, \top, \times\}$. The inequalities are defined using the axioms: $\bot \leq \tau \times \tau', \bot \leq \top, \tau \times \tau' \leq \top$, and whenever $\sigma_1 \leq \tau_1$ and $\sigma_2 \leq \tau_2$ hold we have also $\sigma_1 \times \sigma_2 \leq \tau_1 \times \tau_2$. Note that these rules allow to determine inequality between the types with no variables. Thus it makes sense to consider the following problem.[1]

Definition 3. `(type entailment)`
Input: A finite set $E = \{\sigma_1 \leq \tau_1, \ldots, \sigma_n \leq \tau_n\}$ of inequalities between open types together with an inequality $\sigma \leq \tau$.
Question: Is there a substitution S such that for each $i = 1, \ldots, n$ the inequality $S(\sigma_i) \leq S(\tau_i)$ holds and $S(\sigma) \leq S(\tau)$ does not?

W.l.o.g. we can assume that all the open types have one of the following forms: $\top, \bot, x, x_1 \times x_2$. The type entailment problem with inputs restricted in this way can be reduced to the one-step parallel rewriting theory with a rewriting system that is defined over a signature $\Sigma = \{\top, \bot, f, f_1, f_{L1}, f_{L2}, f_{R1}, \times\}$ where $\top, \bot$ have arity 0, symbols f_* have arity 1 and $\times$ has arity 2. The logic we deal with allows to use object variables only. This requires to provide a way to relate a

[1] This is not the cannonical formulation of the type entailment problem, but it is equivalent in case one is interesed in the (un)decidability of the problem.

variable which represents a term of the form $x_1 \times x_2$ with the variables which represent x_1 and x_2. The symbols $f, f_1, f_{L1}, f_{L2}, f_{R1}$ facilitate this.

The reduction system has the following rules:

$$
\begin{array}{l}
\textbf{Order reductions:} \\
\bot \to x \times y, \qquad \bot \to \top, \qquad x \times y \to \top \\
\textbf{Well-formedness reductions:} \\
f_i(x) \to f_i(x) \quad \text{for } i = 1, L1, L2, R1 \\
f(x_1) \times x_2 \to f(x_1) \times x_2 \qquad x_1 \times f(x_2) \to x_1 \times f(x_2) \\
f \textbf{ on top reductions:} \\
f(x) \to f_1(x) \qquad f_1(x) \to f(x) \\
\textbf{Decomposition reductions:} \\
f(x_1 \times x_2) \to f_{L1}(x_1) \qquad f_{L1}(x) \to f_{L2}(x) \qquad f_{L2}(x) \to f(x) \\
f(x_1 \times x_2) \to f_{R1}(x_1) \qquad f_{R1}(x) \to f(x).
\end{array}
\tag{1}
$$

The rules in the section *Order reductions* simulate the subtype order. The rules in the section *Well-formedness reductions* allow to make sure that a term which encodes a type σ does not contain $f_1, f_{L1}, f_{L2}, f_{R1}$ and that f can occur only at the root position. The rules in f *on top reductions* are exploited to make sure f indeed occurs at the root position so that the encoding of a type σ is $f(\sigma)$. At last the *Decomposition rules* serve to make sure that a term which encodes $x_1 \times x_2$ appropriately decomposes to terms encoding x_1 and x_2.

The existential formula we are interested in here looks as follows:

$$\exists \boldsymbol{X}.\phi_{\text{wf}} \wedge \phi_{\text{ld}} \wedge \phi_{\text{rd}} \wedge \phi_{\text{ent}}$$

where $\boldsymbol{X}$ contains all the variables in the formula. The intent behind the subformulae is the following: ϕ_{wf} ensures that the encodings of the types are well-formed; ϕ_{ld} ensures that the term $(f(t_1 \times t_2))$ which encodes $x_1 \times x_2$ properly decomposes on the left-hand side to a term which corresponds to x_1 $(f(t_1))$; the role of ϕ_{rd} is similar to ϕ_{ld}, but it deals with the right-hand side $(f(t_2))$; at last ϕ_{ent} serves to encode the entailment problem instance at hand.

Let $E \cup \{\sigma \leq \tau\}$ be an instance of the type entailment problem which we are about to translate. Let $x_1, \ldots, x_m$ denote all the variables in the instance. The precise definitions of the subformulae above are the following.

$$
\begin{array}{l}
\phi_{\text{wf}} = \bigwedge_{i=1}^{m} \phi_{\text{wfv}}(x_i, x_i') \wedge \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{m} \phi_{\text{wfv}}(x_{x_i \times x_j}, x'_{x_i \times x_j}) \\
\phi_{\text{wfv}}(x, x') = \neg x \to^p x \wedge x \to^p x' \wedge x' \to^p x \wedge x' \to^p x' \\
\phi_{\text{ld}} = \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{m} \phi_{\text{ldv}}(i, j) \\
\phi_{\text{ldv}}(i, j) = x_{x_i \times x_j} \to^p x_{x_i}^{L1} \wedge \neg x_{x_i}^{L1} \to^p x_{x_i \times x_j} \wedge \\
\qquad\qquad x_{x_i}^{L1} \to^p x_{x_i}^{L2} \wedge \neg x_{x_i}^{L2} \to^p x_{x_i}^{L1} \wedge \\
\qquad\qquad x_{x_i}^{L2} \to^p x_i \wedge \neg x_i \to^p x_{x_i}^{L2} \\
\phi_{\text{rd}} = \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{m} \phi_{\text{rdv}}(i, j) \\
\phi_{\text{rdv}}(i, j) = x_{x_i \times x_j} \to^p x_{x_j}^{R1} \wedge \neg x_{x_j}^{R1} \to^p x_{x_i \times x_j} \wedge \\
\qquad\qquad x_{x_j}^{R1} \to^p x_j \wedge \neg x_j \to^p x_{x_j}^{R1}
\end{array}
$$

$\phi_{\text{ent}} = \overline{\sigma_1} \to^p \overline{\tau_1} \wedge \cdots \wedge \overline{\sigma_n} \to^p \overline{\tau_n} \wedge \neg\overline{\sigma} \to^p \overline{\tau}$
where the types are encoded as follows:
$\overline{x} = x$
$\overline{x \times y} = x_{x \times y}$

Note that in the presentation above the formulae $\phi_{\text{wf}}, \phi_{\text{ld}}, \phi_{\text{rd}}$ are composed of the formulae $\phi_{\text{wfv}}, \phi_{\text{ldv}}, \phi_{\text{rdv}}$ respectively which express the desired property locally for a particular variable. All the properties enforced by the formulae $\phi_{\text{wf}}, \phi_{\text{ld}}, \phi_{\text{rd}}, \phi_{\text{ent}}$ can be expressed by appropriate lemmas. Due to space limits we give here only the formulation of the least obvious lemma which captures the guarantees ensured by the subformula ϕ_{ent}:

Lemma 1. `(rewriting and inequality)`
For each σ, τ and a substitution S we have that $S(\sigma) \leq S(\tau)$ iff $\overline{S}(\overline{\sigma}) \to^p \overline{S}(\overline{\tau})$ in the rewriting system (1) where $\overline{S}(x_i) = f(S(x_i))$ and $\overline{S}(x_{x_i \times x_j}) = f(S(x_i) \times S(x_j))$.

Proof. The proof is by induction over $S(\sigma)$. The details are left to the reader.

We obtain the following theorem:

Theorem 1. *The decidability of the existential theory of the parallel one-step rewriting implies the decidability of the non-structural type entailment problem in case types are built of $\top, \bot, \times$.*

Although this reduction is not conclusive in the light of Theorem 4, this shows that the problems potentially share much of the structure and that a decidability proof for the type entailment problem can give hint on decidable subcases of the theory we deal with here.

4 The undecidability construction

We reduce here the halting problem for a special kind of Turing machines — left-terminal Turing machines — to the validity of existential formulae in the first-order one-step parallel rewriting logic. Thus we obtain our undecidability.

4.1 Left-terminal Turing machines

Let $M = \langle Q, q_I, Q_f, \Sigma, \delta \rangle$ be a deterministic Turing machine (DTM), where Q is the set of its states, q_I is the initial state, Q_f is the set of its final states, Σ is the alphabet of the Turing machine, and $\delta : Q \times \Sigma \rightharpoonup Q \times \Sigma \times D$ is a partial next step function with D being the set of directions $\{L, R, S\}$. We assume a model in which the Turing machine tape extends on demand with a cell that contains 0 when the machine tries to move rightwards while there is no next tape cell.

Definition 4. `(left-terminal Turing machine)`
We say that DTM M is a *left-terminal Turing machine* (LTTM) when

1. $\Sigma = \{0, 1\}$,

2. for each q such that $\delta(q, w) = (q', l, d)$ we have $d \neq S$,
3. the final configuration is reached only at the left end of the tape.

The goal of the restriction (1) is to downsize the number of rewrite rules. The restriction (2) allows to avoid the technical difficulties caused by moves that do not change the machine tape. The restriction (3) allows to check by rewrite rules that the terminal configuration has been reached.

A *configuration* of an LTTM M is a sequence $\rho_1 \cdot (q, l) \cdot \rho_2$ where $\rho_1, \rho_2 \in \Sigma^*, l \in \Sigma$, and $q \in Q$. The *initial configuration* of M is a configuration of the form $(q_I, l) \cdot \gamma$ where q_I is the initial state of M and $l \cdot \gamma$ is the input sequence for the Turing machine.

The *equivalence* $\sim_M$ is the symmetric, reflexive and transitive closure of the $\rightarrow_M$ relation. Let γ_1 be an initial configuration and γ_n a final configuration. A sequence $\gamma_1, \ldots, \gamma_n$ of configurations that is a witness of $\gamma_1 \sim_M \gamma_n$ equivalence is called an *eq-run*. We consider the following equivalence problem

Definition 5. `(the equivalence problem)`
Input: An initial configuration $(q_I, l) \cdot \rho$ of an LTTM M.
Question: Is there a final configuration $(q, l') \cdot \rho'$ where $q \in Q_f$, and $l' \cdot \rho' \in \Sigma^*$, such that $(q_I, l) \cdot \rho \sim_M (q, l') \cdot \rho'$?

We deal with this problem rather than with the reachability problem as it allows us to express that certain term reductions (e.g. $t_3 \rightarrow^p t_{32}$ for terms in Def. 11 are not caused by a move of an LTTM we simulate. The following theorem holds:

Theorem 2. `(the undecidability of the equivalence)`
The equivalence problem for LTTM is undecidable.

Proof. The proof is by a routine technique similar to the proof of undecidability of Thue systems.

4.2 Rewriting and LTTM

We relate here the term rewriting and the existential fragment of the parallel rewriting logic.

Definition 6. `(existential formula)`
Let

$$\begin{aligned}
\phi_{\text{form}} &= x_1 \not\rightarrow^p x_1 \\
\phi_{\text{start}} &= x_1 \rightarrow^p x_3 \wedge x_3 \not\rightarrow^p x_1 \wedge x_3 \rightarrow^p x_3 \wedge \phi_{\text{loop}}(3) \\
\phi_{\text{run}} &= x_1 \not\rightarrow^p x_2 \wedge x_2 \rightarrow^p x_1 \wedge x_2 \rightarrow^p x_2 \wedge \phi_{\text{loop}}(2) \\
\phi_{\text{end}} &= x_3 \rightarrow^p x_2 \wedge x_2 \not\rightarrow^p x_3 \\
\phi_{\text{loop}}(i) &= x_i \rightarrow^p x_{i1} \wedge x_{i1} \rightarrow^p x_i \wedge x_{i1} \rightarrow^p x_{i2} \wedge x_{i2} \rightarrow^p x_i \wedge x_i \not\rightarrow^p x_{i2}
\end{aligned}$$

The resulting existential sentence is:

$$\phi_M = \exists x_1 x_2 x_3 x_{21} x_{22} x_{31} x_{32} . \phi_{\text{form}} \wedge \phi_{\text{start}} \wedge \phi_{\text{run}} \wedge \phi_{\text{end}}.$$

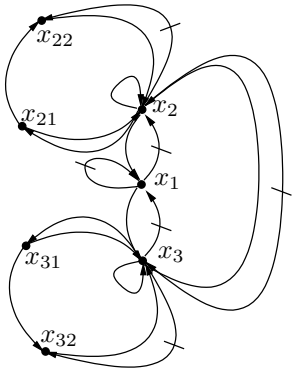

Fig. 1. The relationships between variables in the existential formula

To save the notational burden the parameter of the formula ϕ_{loop} is an index. The formula ϕ_{form} guarantees that the terms we deal with here are appropriate candidates for encodings of an eq-run. The formula ϕ_{start} serves to ensure that a simulation of the LTTM starts with a starting configuration of the machine, ϕ_{end} to ensure that the simulation ends with a final configuration of the machine, and at last ϕ_{run} serves to guarantee that the simulation will obey the transition rules of the machine. The dependencies between the variables $x_1, x_2, x_3, x_{21}, x_{22}, x_{31}, \ldots, x_{32}$ encoded by ϕ_M are presented on Fig. 1.

Let us fix an LTTM M. Based on that we construct a signature Σ_M, a rewriting system R_M and a sentence ϕ_M such that M halts iff ϕ_M holds in $\mathcal{A}_{R_M}$.

Definition 7. `(signature)`
Now, we define the signature Σ_M. It contains the following symbols

$$f, g \text{ of arity } 2, \qquad \bot_g, \bot_g^1, \bot_g^2, \bot_g^3 \text{ of arity } 1, \qquad \bot_f, 0, 1 \text{ of arity } 0 \quad .$$

Additionally, we use elements of the set $Q \times \{0, 1\}$ as symbols of arity 0 where Q is the set of states of M.

Note that the restriction (1) from Def. 4 allows to use here the symbols $0, 1$ only. The intent of the formula ϕ_M is to enforce that the term t_1 substituted on x_1 contains an encoded eq-run of the LTTM we are interested in. Furthermore, it ensures that the term t_3 substituted on x_3 will be almost the same with one exception that $\bot_g$ is replaced with $\bot_g^1$. The formula $\phi_{\text{loop}}(3)$ guarantees roughly that $\bot_g^1$ actually occurs there. The term t_2 substituted on x_2 also contains the symbol which is guaranteed by $\phi_{\text{loop}}(2)$. The loop between x_1, x_3 and x_2 guarantees that t_2 differs from t_3 so that the terminal configuration at the top of t_3 is deleted from t_2. In this way, we obtain the opportunity to compare subsequent machine moves in the fashion presented on Fig. 2.

Now, we are able to define an encoding of a configuration and an eq-run.

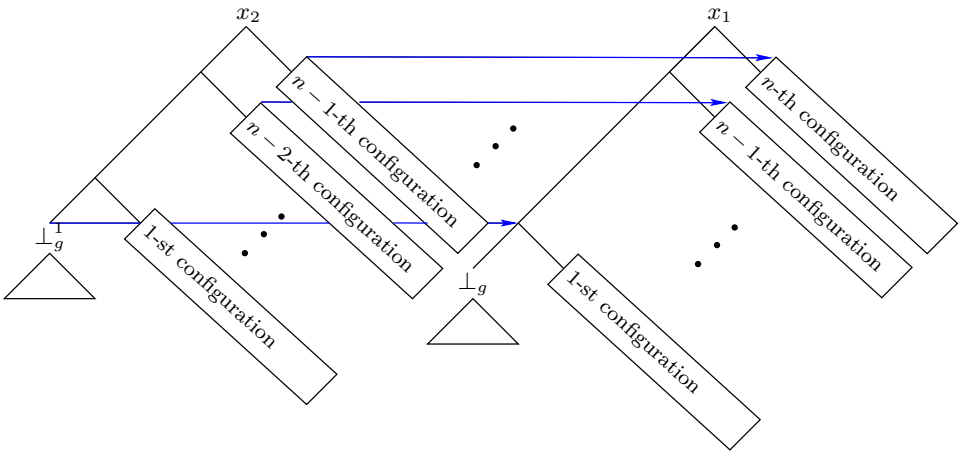

Fig. 2. The mechanism for the next step simulation.

Definition 8. `(encoding of configurations)`
Let ρ be a configuration of M. We define the encoding $\mathrm{enc}(\rho)$ of the configuration in the term algebra over the signature from Def. 7 as follows:

- $\mathrm{enc}(\varepsilon) = \perp_f$;
- $\mathrm{enc}(l \cdot \rho') = f(l, \mathrm{enc}(\rho'))$ for $l \in \{1, 0\}$;
- $\mathrm{enc}((q, l) \cdot \rho') = f(\langle q, l\rangle, \mathrm{enc}(\rho'))$ where $l \in \{1, 0\}$.

Definition 9. `(encoding of an eq-run)`
Let $\rho_1, \ldots, \rho_n$ be an eq-run of M. The term $\mathrm{e2t}(\rho_1, \ldots, \rho_n)$ is defined as:

- $\mathrm{e2t}(\rho_1) = g(\perp_g(t), t)$, where $t = \mathrm{enc}(\rho_1)$;
- $\mathrm{e2t}(\rho_1, \ldots, \rho_i) = g(\mathrm{e2t}(\rho_1 \ldots, \rho_{i-1}), \mathrm{enc}(\rho_i))$.

Given an LTTM M and its initial state q_I together with the input word $l \cdot \rho_1$, we can define a rewriting system which has the behaviour equivalent to the behaviour of M on $l \cdot \rho_1$.

Definition 10. `(rewrite system)`
The rewrite rules can be divided as follows:

1. The rules relevant to the detection of the starting configuration:

$$\perp_g(\mathrm{enc}((q_I, l) \cdot \rho_1)) \to \perp_g^1(\mathrm{enc}((q_I, l) \cdot \rho_1)),$$
$$\perp_g^1(x) \to \perp_g^1(x), \quad \perp_g^1(x) \to g(\perp_g(x), x)$$

   where $(q_I, l) \cdot \rho_1$, as above, is the initial configuration of M.
2. The rules relevant to detection of the final configuration:

$$g(g(x, y), f(\langle q, l\rangle, z)) \to g(x, y)$$

   where l ranges over $\{0, 1\}$, and $q \in Q_f$ (Q_f contains the final states of M).

3. The rules that check the form of an eq-run:

$$
\begin{array}{ll}
f(f(x,y),z) \to f(f(x,y),z), & g(x,g(y,z)) \to g(x,g(y,z)),\\
f(g(x,y),z) \to f(g(x,y),z), & f(x,g(y,z)) \to f(x,g(y,z)),\\
g(f(x,y),z) \to g(f(x,y),z), & g(x,\bot_g(y)) \to g(x,\bot_g(y)),\\
f(\bot_g(x),y) \to f(\bot_g(x),y), & f(x,\bot_g(y)) \to f(x,\bot_g(y)),\\
g(c_1,x) \to g(c_1,x), & g(x,c_2) \to g(x,c_2),\\
\bot_g(\bot_g(x)) \to \bot_g(\bot_g(x)), & \bot_g(g(x,y)) \to \bot_g(g(x,y)),\\
\bot_g(c_2) \to \bot_g(c_2), & f(x,c_2) \to f(x,c_2),\\
f(\bot_f,x) \to f(\bot_f,x), & \bot_g^i(x) \to \bot_g^i(x) \quad \text{for } i=1,2,3,\\
\multicolumn{2}{c}{g(g(x,f(\langle q,l\rangle,y)),f(\langle q,l\rangle,z)) \to g(g(x,f(\langle q,l\rangle,y)),f(\langle q,l\rangle,z))}
\end{array}
$$

   where c_1 ranges over all symbols of arity 0, c_2 over all symbols of arity 0 except from $\bot_f$, l ranges over $\{0,1\}$, and q ranges over Q_f.
4. The rules which check that an eq-run obeys the state transitions: all the rules of the form

$$
\begin{array}{l}
f(\langle q_1,l_1\rangle, f(l_2,x)) \to f(l_3, f(\langle q_2,l_2\rangle,x)),\\
f(\langle q_1,l_1\rangle, \bot_f) \to f(l_3, f(\langle q_2,0\rangle,\bot_f))
\end{array} \tag{2}
$$

   in case $\delta(q_1,l_1) = (q_2,l_3,R)$; and rules of the form

$$
f(l_1, f(\langle q_1,l_2\rangle,x)) \to f(\langle q_2,l_1\rangle, f(l_3,x)), \tag{3}
$$

   in case $\delta(q_1,l_2) = (q_2,l_3,L)$ (note that this is the place where the point 2 of the definition of LTTM is used). Additionally, all the rules of the form $t_1 \to t_2$ where $t_2 \to t_1$ occurs in (2) or (3) above.
5. The rules that allow to have loops defined by $\phi_{\text{loop}}(i)$:

$$
\bot_g^i(x) \to \bot_g^{i'}(x) \quad \bot_g^2(x) \to \bot_g^1(x)
$$

   where $i = 1,2,3$ and $i' = (i+1 \mod 3)+1$.

Now that we see the rewriting system, we can prove one direction of the equivalence between the rewriting and the LTTMs.

Definition 11. (`terms that satisfy` ϕ_M)
We can now define the terms $t_1, t_2, t_3, t_{31}, t_{32}, t_{21}, t_{22}$ which are used to satisfy the formula ϕ_M:

- $t_1 = \text{e2t}(\rho_1,\ldots,\rho_n)$,
- $t_2 = \text{e2t}(\rho_1,\ldots,\rho_{n-1})[\gamma \leftarrow \bot_g^1(t)]$ where $\text{e2t}(\rho_1,\ldots,\rho_{n-1})|_\gamma = \bot_g(t)$,
- $t_3 = t_1[\gamma \leftarrow \bot_g^1(t)]$ where $t_1|_\gamma = \bot_g(t)$,
- $t_{21} = t_2[\gamma \leftarrow \bot_g^2(t)]$ where γ is such that $t_2|_\gamma = \bot_g^1(t)$,
- $t_{22} = t_2[\gamma \leftarrow \bot_g^3(t)]$ where γ is such that $t_2|_\gamma = \bot_g^1(t)$,
- $t_{31} = t_3[\gamma \leftarrow \bot_g^2(t)]$ where γ is such that $t_3|_\gamma = \bot_g^1(t)$,
- $t_{32} = t_3[\gamma \leftarrow \bot_g^3(t)]$ where γ is such that $t_3|_\gamma = \bot_g^1(t)$.

Lemma 2. (from runs of machines to witnesses of the formula)
If $\rho_1, \ldots, \rho_n$ is a run of an LTTM M then terms $t_1, t_2, t_3, t_{21}, t_{22}, t_{31}, t_{32}$ from Def. 11 substituted for $x_1, x_2, x_3, x_{21}, x_{22}, x_{31}, x_{32}$ respectively witness that the formula ϕ_M holds.

Proof. The proof is by a routine case analysis.

The reduction $t_1 \rightarrow^p t_1$ is guaranteed not to hold as the only rules which allow the self-rewriting are in the group [3] and in the third rule of the group [1] while none of the patterns in the left-hand sides of the rules occur in t_1. This proves that ϕ_{form} holds.

For the reductions in ϕ_{start}, the reduction $t_1 \rightarrow^p t_3$ is guaranteed by the first rule in the group [1]. The reduction $t_3 \rightarrow^p t_3$ is guaranteed by the second rule in the same group. The reduction $t_3 \rightarrow^p t_1$ is impossible as the only way to dispose of the symbol $\bot_g^1$ is either to use the third rule in [1], but then the number of gs must increase, or to use the rule in [5], but using this you cannot obtain $\bot_g$. The rewrites in the subformula $\phi_{\text{loop}}(3)$ are possible because of the rules in [5]. The rewrite $t_3 \rightarrow^p t_{32}$ is impossible as there is no rule which exchanges $\bot_g^1$ with $\bot_g^3$. Thus ϕ_{start} holds.

For the reductions in ϕ_{run}, the reduction $t_1 \rightarrow^p t_2$ is impossible for the same reasons as in the case of the reduction $t_3 \rightarrow^p t_1$. The reduction $t_2 \rightarrow^p t_1$ is possible because $t_2|_{(1^i)2}$ represents the configuration of M one step before the configuration represented by $t_1|_{(1^i)2}$ and the rules in the group [4] can be applied at an appropriate position in t_2 to obtain the corresponding part of t_1. Moreover, the third rule in [1] can be used to dispose of $\bot_g^1$. The rewrites in the subformula $\phi_{\text{loop}}(2)$ hold for the reasons similar to the case of $\phi_{\text{loop}}(3)$. Thus ϕ_{run} holds.

For the reductions in ϕ_{end}, the reduction $t_3 \rightarrow^p t_2$ is possible by the rules in [2] while the reduction $t_2 \rightarrow^p t_3$ is impossible as the only rule which increases the number of occurences of the symbol g is the third rule in [1], but this rule can be applied only at one place in t_2 (as there is only one occurrence of $\bot_g^1$), but then it is impossible to obtain t_3 as t_3 contains $\bot_g^1$ which is removed by the rule. Thus ϕ_{end} holds. The details are left for the reader.

The lemma above establishes one direction of the relation between M and ϕ_M. In order to establish the other direction we have to define several forms of terms which can be enforced by certain graph structures expressed in the formula. The first set of forms of interest are the forms which can be defined by means of the formula $x \not\rightarrow^p x$ (i.e. forms where certain local patterns are forbidden) and which are close enough to the encoding of the configuration and run. In addition we define several forms which are similar and are used in the proof for technical reasons.

Definition 12. (pseudo-configuration form)
The set of terms in the *pseudo-configuration* form is defined inductively as the smallest set that contains terms: $\bot_f$, and $f(c, t)$ where $c \in Q \times \{0, 1\} \cup \{0, 1\}$ and t is in the pseudo-configuration form.

Definition 13. (pseudo-run form)
The set of terms in the *pseudo-run* form is defined inductively as the smallest set that contains terms:

1. $\bot_g(t)$ where t is in the pseudo-configuration form,
2. $g(t_1, t_2)$ where the subterm t_1 is in the pseudo-run form and t_2 is in the pseudo-configuration form

and in which the pattern $g(g(x, f(\langle q, l\rangle, y)), f(\langle q, l\rangle, z))$ does not occur.

The terms in the *weak pseudo-run form* are defined as above, but the final requirement about the pattern is dropped. A term is in the *(weak) $\bot_g^1$-pseudo-run* form when after replacing symbols $\bot_g^1$ with $\bot_g$ it is in the (weak) pseudo-run form.

Definition 14. (x_1-form and x_*-form)
We say that a term t is in the *x_1-form* when it is either a constant, or is in the pseudo-configuration form, or is in the pseudo-run form.

A term t is in the *x_*-form* when it is either in the x_1-form or is in the weak $\bot_g^1$-pseudo-run form.

We have now a bunch of technical lemmas that relate the topological structures in the formula ϕ_M with the forms defined above. The proofs of these lemmas are by a routine case analysis combined with induction. The details of the proofs are in Appendix A.

We start with a characterisation of terms that cannot be reduced to themselves. This is expressed by the subformula ϕ_{form} of ϕ_M.

Lemma 3. (formula and the pseudo-run form)
A term t satisfies the formula $x \not\to^p x$ iff t is in x_1-form.

The following lemma characterises both the reduction in formulae $\phi_{\text{loop}}(i)$ for $i = 2, 3$ and the reductions between x_1, x_3 and x_2.

Lemma 4. (loop and the pseudo-configuration form)
If t is in the pseudo-configuration form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula

$$x_* \to^p x_{*1} \wedge x_{*1} \to^p x_{*2} \wedge x_{*2} \to^p x_* \wedge x_* \not\to^p x_{*2}.$$

The following lemma characterises the reductions of pseudo-configuration and weak pseudo-run terms.

Lemma 5. (reduction of the pseudo-configuration and -run forms)
(1) Let t_1 be in the pseudo-configuration form. If $t_1 \to^p t_2$ then t_2 is in the pseudo-configuration form.
(2) Let t_1 be in the weak pseudo-run form. If $t_1 \to^p t_2$ then t_2 is in the weak pseudo-run form or in the weak $\bot_g^1$-pseudo-run form.

The following lemma allows to enforce that the reduction $x_1 \to^p x_3$ results in a term substituted for x_3 which contains $\bot_g^1$.

Lemma 6. (reduction in a loop)
If t is in the weak pseudo-run form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula $\phi_{\text{loop}}(*)$.

The following lemma allows to enforce that the redution $x_2 \to^p x_3$ does not lose $\bot^1_g$.

Lemma 7. (on x_*-form)
Let $t_1 \to^p t_2$. If t_1 is in the x_*-form and there are t_{21} and t_{22} such that the substitution $S = \{x_* := t_2, x_{*1} := t_{21}, x_{*2} := t_{22}\}$ satisfies the formula $\phi_{\text{loop}}(*)$ then t_2 is in the weak $\bot^1_g$-pseudo-run form.

This lemma allows to construct a run of M based on the witnesses for ϕ_M.

Lemma 8. (from witnesses of the formula to runs of machines)
Let R be a rewriting system from Def. 10 constructed for an LTTM M. If the formula ϕ_M from Def. 6 holds for the rewriting with R, then there is an eq-run of M.

Proof. Let $t_1, t_2, t_3, t_{21}, t_{22}, t_{31}, t_{32}$ be the witnesses that the formula ϕ_M holds. Note that the terms t_1, t_2 and t_3 are pairwise different. Since $t_1 \not\to^p t_1$, we obtain by Lemma 3 that t_1 is in the pseudo-run form or pseudo-configuration form, or is a constant. It cannot be a constant since we have $t_1 \to^p t_3$ and there is no rewriting rule with a constant on the left-hand side. Moreover, t_1 cannot be in the pseudo-configuration form as this contradicts Lemma 4 applied to t_1, t_2, and t_3. Thus, t_1 may only be in the pseudo-run form (*).

Lemma 7 implies that t_2 and t_3 are in the weak $\bot^1_g$-pseudo-run form. This implies that the reduction $t_1 \to^p t_3$ must be done with the use of the first rule in the group [1] of Def. 10. This together with (*) implies that the reduction $t_2 \to^p t_1$ must use the third rule in the group [1]. Since the third rule in [1] increases the number of gs on the leftmost branch, one of the rewrites $\to^p t_2 \to^p t_1$ must decrease it. The only rules that can decrease the number of gs are the rules from [2]. This, however, can be used only in the rewriting $t_3 \to^p t_2$ as otherwise the use of the third rule from [1] in $t_2 \to^p t_1$ is impossible. Let γ be the address where a rule from [2] is applied to t_3. Let $\gamma \cdot 1^n 2(t_i) = u^n_i$ where $i = 1, 2, 3$. The terms u^n_i are in the pseudo-configuration form (as u^n_i are and by Lemma 5). This means that only the rules in the group [4] can be applied to them. Additionally, we have the dependencies $u^i_1 \to^p_* u^i_3 = u^{i-1}_2 \to^p_* u^{i-1}_1$ ($u^i_3 = u^{i-1}_2$ as a rule is applied at γ in $t_3 \to^p t_2$). This justifies that $u^i_1 \to^p_* u^{i-1}_1$. Let $\gamma \cdot 0^{m+1}$ be the address of $\bot^1_g$ in t_3. Note that u^m_1 is the encoding of an initial configuration, as the first rule from [1] is used in $t_1 \to^p t_3$. The term u^m_1 is a properly formed encoding of a configuration. By induction on i we obtain that all pseudo-configuration forms u^{m-i}_1 are proper encodings as well as the rules in [4] directly encode forward and backward transitions of M. Further, we obtain by the downwards induction on i that each u^i_1 is in the relation $\sim_M$ with u^m_1. The application of a reduction from [2] in $t_3 \to^p t_2$ enforces that u^0_3 represents a final configuration of M by the point (3) of Def. 4. In this way, we obtain from a solution of the formula ϕ_M two configurations ρ_1, ρ_2 such that ρ_1 is the initial one, ρ_2 is the final one and $\rho_1 \sim_M \rho_2$ holds.

As the result of this lemma we obtain the theorem:

Theorem 3. `(translation between machines and rewrite systems)` *For each LTTM M and its initial configuration ρ there is a rewrite system R such that $\rho \sim_M \rho'$ for some final configuration ρ' iff the formula ϕ_M holds.*

Theorem 4. `(the undecidability of one-step parallel rewriting theory)`
There is no algorithm that given a signature Σ, an existential formula ϕ, and a rewrite system R over Σ can decide if ϕ is satisfied in the parallel rewriting structure $\mathcal{A}_R$ from Def. 1. In particular, the existential fragment of the theory of one-step parallel rewriting is undecidable.

5 Discussion

The use rewrite rules of the form $t \to t$ in [Tre98] has been criticised. The current paper also uses such rules. It is an open question if this can be avoided here.

Vorobyov in [Vor02] distinguishes weak and strong undecidability for this kind of theory. The strong one holds when the undecidability is proved for a fixed rewriting system while weak holds in other cases. In the sense, this paper presents a weak undecidability. However, the current construction works with a fixed formula. I conjecture that the strong version in the Vorobyov's sense also holds for the existential fragment of the theory. I did not prove this strong version as the plans of such constructions that I could develop were very complicated.

The paper [CSTT99] uses a one-step rewriting theory with $\to$ predicate replaced by predicates $\to^r$, one for each rewriting rule r. The main result of the current paper can be straightforwardly adapted to this version of the theory even in case of Vorobyov's strong sense.

References

[AK96] Ilies Alouini and Claude Kirchner. Toward the concurrent implementation of computational systems. In *ALP '96: Proceedings of the 5th International Conference on Algebraic and Logic Programming*, pages 1–31, London, UK, 1996. Springer-Verlag.

[Ass00] Uwe Assmann. Graph rewrite systems for program optimization. *ACM Trans. Program. Lang. Syst.*, 22(4):583–637, 2000.

[CSTT99] Anne-Cécile Caron, Franck Seynhaeve, Sophie Tison, and Marc Tommasi. Deciding the satisfiability of quantifier free formulae on one-step rewriting. In Paliath Narendran and Michael Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *LNCS*, pages 103–117. Springer-Verlag, 1999.

[GKM87] Joseph Goguen, Claude Kirchner, and José Meseguer. Concurrent term rewriting as a model of computation. In *Proc. of a workshop on Graph reduction*, pages 53–93, London, UK, 1987. Springer-Verlag.

[HR97] Fritz Henglein and Jakob Rehof. The complexity of subtype entailment for simple types. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 352. IEEE Computer Society, 1997.

[HR98] Fritz Henglein and Jakob Rehof. Constraint automata and the complexity of recursive subtype entailment. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 616–627. Springer-Verlag, 1998.

[Jac96] Florent Jacquemard. *Automates d'arbres et Réécriture de termes*. PhD thesis, Université de Paris-Sud, 1996.

[LR99] Sébastien Limet and Pierre Réty. A new result about the decidability of the existential one-step rewriting theory. In Paliath Narendran and Michael Rusinowitch, editors, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *LNCS*, pages 118–132. Springer-Verlag, 1999.

[Mar97] Jerzy Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In Hubert Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, number 1232 in LNCS. Springer-Verlag, June 1997.

[MOM02] Narciso Martí-Oliet and José Meseguer. Rewriting logic: roadmap and bibliography. *Theor. Comput. Sci.*, 285(2):121–154, 2002.

[NP99] Joachim Niehren and Tim Priesnitz. Entailment of non-structural subtype constraints. In *Proceedings of the 5th Asian Computing Science Conference on Advances in Computing Science*, pages 251–265. Springer-Verlag, 1999.

[NP03] Joachim Niehren and Tim Priesnitz. Non-structural subtype entailment in automata theory. *Inf. Comput.*, 186(2):319–354, 2003.

[NPR97] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification and one-step rewriting. In William McClune, editor, *14th International Conference on Automated Deduction*, number 1249 in LNAI, pages 34–48. Springer-Verlag, 1997.

[STT97] Franck Seynhaeve, Marc Tommasi, and Ralf Treinen. Grid structures and undecidable constraint theories. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, number 1214 in LNCS, pages 357–368. Springer-Verlag, 1997.

[STTT01] Franck Seynhaeve, Sophie Tison, Marc Tommasi, and Ralf Treinen. Grid structures and undecidable constraint theories. *Theoretical Computer Science*, 1–2(258):453–490, May 2001.

[Tis90] Sophie Tison. Automates comme outil de décision dans les arbres. Dossier d'habilitation à diriger des recherches, December 1990. In French.

[Tre96] Ralf Treinen. The first-order theory of one-step rewriting is undecidable. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA'96)*, number 1103 in LNCS, pages 276–286. Springer-Verlag, July 1996.

[Tre98] Ralf Treinen. The first-order theory of linear one-step rewriting is undecidable. *Theoretical Computer Science*, 1–2(208):149–177, November 1998.

[Vor97] Sergei Vorobyov. The first-order theory of one step rewriting in linear noetheran systems is undecidable. In Hubert Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, number 1232 in LNCS, pages 254–268. Springer-Verlag, June 1997.

[Vor02] Sergei Vorobyov. The undecidability of the first-order theories of one step rewriting in linear canonical systems. *Inf. Comput.*, 174:1–32, 2002.

A Proofs of lemmas

We present here the proofs of the lemmas omitted from the main text.
Proof of Lemma 3:
Let us recall the formulation of the lemma first.

A term t satisfies the formula $x \not\rightarrow^p x$ iff t is in the x_1-form.

First note that when t is in the x_1-form then it is either in the pseudo-configuration form, or in the pseudo-run form, or is one of the symbols of arity 0.

($\Rightarrow$) Induction on the structure of t.

(A) If t is a constant then the result is trivial.

(B) If $t = \bot_g(t')$ then as t' also satisfies the formula $x \not\rightarrow^p x$ we can conclude by induction hypothesis that t' is in the pseudo-configuration form, or in the pseudo-run form, or is one of the symbols of arity 0. The term t' cannot be in the pseudo-run form as R contains the rules $\bot_g(g(x,y)) \rightarrow \bot_g(g(x,y))$ and $\bot_g(\bot_g(x)) \rightarrow \bot_g(\bot_g(x))$. Since the rules $\bot_g(c_3) \rightarrow \bot_g(c_3)$ for all $c_3 \in \Sigma^0_M \backslash \{\bot_f\}$ are in R, the only constant that can be equal to t' is $\bot_f$. This means that t' is in the pseudo-configuration form which in turn implies that t is in the pseudo-run form.

(C) The situation that $t = \bot^i_g(t')$ for $i = 1, \ldots, 3$ is impossible since R contains the rules $\bot^i_g(x) \rightarrow \bot^i_g(x)$ for $i = 1, \ldots, 3$.

(D) If $t = f(t_1, t_2)$ then as t_i for $i = 1, 2$ also satisfy the formula $x \not\rightarrow^p x$ we can conclude by induction hypothesis that each t_i is in the pseudo-configuration form, or in the pseudo-run form, or is a constant. The term t_1 cannot be in the pseudo-run or pseudo-configuration form as R contains the rules $f(f(x,y),z) \rightarrow f(f(x,y),z)$, $f(\bot_f, x) \rightarrow f(\bot_f, x)$, $f(g(x,y),z) \rightarrow f(g(x,y),z)$, $f(\bot_g(x),y) \rightarrow f(\bot_g(x),y)$. Thus, t_1 can only be a constant different than $\bot_f$ which means $t_1 \in \{0,1\} \cup Q \times \{0,1\}$. The term t_2 must be in the pseudo-configuration form as R contains the rules $f(x, g(y,z)) \rightarrow f(x, g(y,z))$, $f(x, \bot_g(y)) \rightarrow f(x, \bot_g(y))$, and $f(x, c_3) \rightarrow f(x, c_3)$ where $c_3 \in \Sigma^0_M \backslash \{\bot_f\}$. This verifies that t is in the pseudo-configuration form.

(E) If $t = g(t_1, t_2)$ then the proof is similar to the previous case. We use the rules $g(f(x,y),z) \rightarrow g(f(x,y),z)$, $g(c_1, x) \rightarrow g(c_1, x)$ where $c_1 \in \Sigma^0_M$ to check the form of t_1 and $g(x, g(y,z)) \rightarrow g(x, g(y,z))$, $g(x, c_2) \rightarrow g(x, c_2)$ where $c_2 \in \Sigma^0_M \backslash \{\bot_f\}$, $g(x, \bot_g(y)) \rightarrow g(x, \bot_g(y))$ to check the form of t_2. The last rule in [3] guarantees that t is in the pseudo-run form.

($\Leftarrow$) The proof in this direction is by induction on t.

(A) If t is a constant then the result is trivial as there is no rule $l \rightarrow r$ in Def. 10 in which l is a constant.

(B) If $t = \bot_g(t')$ then t may only be in the pseudo-run form and then t' is in the pseudo-configuration form. By the induction hypothesis $t' \not\rightarrow^p t'$. Thus, the only possible redex is at the root address of t. This is impossible though, as there is no reduction rule with the left-hand side equal to either $\bot_g(\bot_f)$ or $\bot_g(f(s_1, s_2))$ where s_1, s_2 are terms, and as the only rule (the first rule in the group [1]) with the left-hand side equal to $\bot_g(x)$ has different right-hand side.

(C) The cases when $t = \bot_g^i(t')$ are impossible as t of this form cannot be a constant or a term in the pseudo-configuration form or a term in the pseudo-run form.

(D) If $t = f(t_1, t_2)$ then t is in the pseudo-configuration form. This implies that $t_1 \in Q \times \{0,1\} \cup \{0,1\}$ and t_2 is in the pseudo-configuration form. We have $t_1 \not\to^p t_1$ as t_1 is a constant. Additionally, $t_2 \not\to^p t_2$ holds by induction hypothesis. This means that the only possible redex for $t \to^p t$ is at the root address of t. Since $t_1 \in Q \times \{0,1\} \cup \{0,1\}$, the only possible rule is $f(x, c_2) \to f(x, c_2)$ from the point [3] of Def. 10. The reduction is impossible as $c_2 \in \Sigma_M^0 \backslash \{\bot_f\}$ and as the corresponding t_2 may be only $\bot_f$ (t is in the pseudo-configuration form).

(E) If $t = g(t_1, t_2)$ then t may only be in the pseudo-run form. This implies that t_1 is in the pseudo-run form and t_2 is in the pseudo-configuration form. The induction hypothesis for both t_1 and t_2 implies that $t_1 \not\to^p t_1$ and $t_2 \not\to^p t_2$ Thus, the only possible redex address is at the root of t. This means that only the rules from the points [2]-[3] can be applied. The rules in the point [3] cannot be applied because it would imply that t_1 is not in the pseudo-run form or t_2 is not in the pseudo-configuration form, or that t is in the weak pseudo-run form which is not the pseudo-run form. The applicability of the rules in [2] would mean that $t \neq t$. □

Proof of Lemma 4:
Let us recall the formulation of the lemma first.

> If t is in the pseudo-configuration form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula:
>
> $$x_* \to^p x_{*1} \wedge x_{*1} \to^p x_{*2} \wedge x_{*2} \to^p x_* \wedge x_* \not\to^p x_{*2}. \qquad (4)$$

Suppose, there are t_1, t_2 such that the substitution S satisfies the formula (4). Note that if t, t_1 and t_2 satisfy the formula then they are pairwise different. Otherwise we obtain contradiction as one pair should satisfy both $x \to^p y$ and $x \not\to^p y$. We continue by induction on the term t.

(A) $t = \bot_f$. This situation is impossible as there are no rules with $\bot_f$ on the left-hand side.

(B) $t = f(c, s)$ where $c \in Q \times \{0,1\} \cup \{0,1\}$ and s is in the pseudo-configuration form. The rewriting $t_2 \to^p t$ cannot be performed with the use of the rules in the group [1] as the right-hand sides there contain g or $\bot_g^1$ and terms in the pseudo-configuration form cannot contain these symbols. The rewriting cannot be performed with the use of the rules in [2] either, as g does not occur in t. The rules from [3] cannot be used as $t_1 \neq t_2$ (in case the rules are used together with other ones we may assume they are not used as left-hand sides are equal to the right-hand ones). The rules in [5] cannot be used as there are no $\bot_g^i$ for $i = 1, 2, 3$ in t. The only remaining case is that only rules in [4] are used in $t_2 \to t$. This is, however, impossible, as the rules in [4] are reversible. (Note that this is a place where we exploit the fact that we use the equivanece $\sim_M$.) □

Proof of Lemma 5:
Let us recall the formulation of the lemma first.

(1) Let t_1 be in the pseudo-configuration form. If $t_1 \to^p t_2$ then t_2 is in the pseudo-configuration form.
(2) Let t_1 be in the weak pseudo-run form. If $t_1 \to^p t_2$ then t_2 is in the weak pseudo-run form or in the weak $\bot_g^1$-pseudo-run form.

• The proof of (1). Induction on the term t_1.

(A) $t_1 = \bot_f$. As there are no rules with $\bot_f$ on the left-hand side this case is impossible.

(B) $t_1 = f(c, t_1')$ where $c \in Q \times \{0,1\} \cup \{0,1\}$ and t_1' is in the pseudo-configuration form. We have two subcases here.

(i) The reduction $t_1 \to^p t_2$ involves a redex in the root. The rules in groups [1], [2], [5] in Def. 10 and most of the rules in the group [3] cannot be used as their left-hand sides do not start with f. The remaining rules of the group [3] cannot be used as c is a constant or because t_1 is in the pseudo-configuration form. If a rule in the group [4] is used then t_2 remains in the pseudo-configuration form. Thus we can conclude that t_2 is in the pseudo-configuration form in this subcase.

(ii) The reduction $t_1 \to^p t_2$ does not involve a redex in the root. We have then $t_2 = f(c, t_2')$. In this case the only possible reduction is as $t_1' \to^p t_2'$ since c is a constant (there are no reductions with constants as left-hand sides). The induction hypothesis gives here that t_2' is in the pseudo-configuration form and by the definition of the pseudo-configuration form we have that $t_2 = f(c, t_2')$ is in the pseudo-configuration form as well.

• The proof of (2). Induction on the term t_1.

(A) $t_1 = \bot_g(t_1')$ where t_1' is in the pseudo-configuration form. We have two subcases here.

(i) In case the reduction is at the root address, we can only apply the first rule in the group [1] from Def. 10 (the rules in the group [3] cannot be applied as this would imply t_1 is not in the pseudo-run form, the rules in other groups have no $\bot_g$ in the left-hand sides). Then $t_2 = \bot_g^1(t_1')$ which is a term in the weak $\bot_g^1$-pseudo-run form.

(ii) In case the reduction is not at the root address we have that $t_2 = \bot_g(t_2')$ and $t_1' \to^p t_2'$. The claim (1) of the current lemma implies that t_2' is in the pseudo-configuration form as well. It immediately follows, then, that t_2 is in the weak pseudo-run form.

(B) $t_1 = g(t_1', t_1'')$ where t_1' is in the pseudo-run form and t_1'' in the pseudo-configuration form. We have two subcases here.

(i) The reduction $t_1 \to^p t_2$ involves a redex in the root. The rules in groups [1], [4], and [5] in Def. 10 and most of the rules in the group [3] cannot be used as their left-hand sides do not start with g. The remaining rules of the group [3] cannot be used as t_1 is in the pseudo-run form. If a rule in the group [2] is used then $t_2 = t_1'$ while t_1' is in the pseudo-run form.

(ii) The reduction $t_1 \to^p t_2$ does not involve a redex in the root. We have then $t_2 = g(t_2', t_2'')$ where $t_1' \to_*^p t_2'$ and $t_1'' \to_*^p t_2''$ and at least one of $t_1' \to^p t_2'$ and $t_1'' \to^p t_2''$ holds. By the case (1) of the current lemma or because $t_1'' = t_2''$ we have that t_2'' is in the pseudo-configuration form. Similarly, by induction hypothesis

or because $t'_1 = t'_2$ we obtain that t'_2 is in the pseudo-run form or weak $\bot^1_g$-pseudo-run form. These two facts are enough to conclude that t_2 is in the weak $\bot^1_g$-pseudo-run form. □

Proof of Lemma 6:
Let us recall the formulation of the lemma first.

> If t is in the weak pseudo-run form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula $\phi_{\text{loop}}(*)$.

Suppose s satisfies $\phi_{\text{loop}}(*)$. We analyse here the reductions between t, t_1 and t_2. First note that t, t_1, t_2 must be pariwise different as otherwise we obtain two terms (not necessarily different) for which both $x \to^p x'$ and $x \not\to^p x'$ is required. In this light, we may assume the rules from the group [3] are not used in the reductions among the terms as their left-hand sides are equal to the right-hand ones.

The reductions in the groups [1], [2], [5] are not reversible in one step, or are not applicable at all. So the reduction $t \to^p t_1$ must be done with the rules from the group [4]. This means that t_1 is in the weak pseudo-run form too.

We can analyse then the reduction $t_1 \to^p t_2$ now. As t_1 is in the weak pseudo-run form, the only possible reduction from the group [1] is the first one. This rule, however, introduces $\bot^1_g$ and the only way to get rid of $\bot^1_g$ is either by the third rule in [1] or by the rule $\bot^1_g(x) \to \bot^2_g(x)$. The former rule introduces additional g and, in order to dispose of both g and $\bot^1_g$, one has to use more than one step so it is impossible to get back to t. Similarly, the latter rule introduces $\bot^2_g$ which does not occur in t_2 and in order to replace $\bot^2_g$ with $\bot_g$ one has to use more than one step.

The rules from [2] in the reduction $t_1 \to^p t_2$ would decrease the number of occurrences of g. This can only be compensated by the use of the third rule in [1]. Then one has to use yet another step to introduce $\bot^1_g$ and it is impossible to accomplish both these steps before t_2 is reached.

The rules in [4] cannot be used as then we arrive with a weak pseudo-run form in t_2 and the only way to obtain t back is to use rules in [4] again and this is contradictory with the fact that $t \not\to^p t_2$ and that the rules in [4] are reversible. (Note that this is another place where we exploit the fact that we use the equivanece $\sim_M$.) □

Proof of Lemma 7:
Let us recall the formulation of the lemma first.

> Let $t_1 \to^p t_2$. If t_1 is in the x_*-form and there are t_{21} and t_{22} such that the substitution $S = \{x_* := t_2, x_{*1} := t_{21}, x_{*2} := t_{22}\}$ satisfies the formula $\phi_{\text{loop}}(*)$ then t_2 is in the weak $\bot^1_g$-pseudo-run form.

First note that if t_2, t_{21} and t_{22} satisfy the formula $\phi_{\text{loop}}(*)$ then they are pairwise different as otherwise we obtain contradiction because for one pair we have both $x \to^p y$ and $x \not\to^p y$ satisfied. The rest of the proof is by induction on the term t_1.

(A) t_1 is a constant. There are no rules with a constant on the left-hand side so the case is impossible.

(B) $t_1 = f(s_1, s_2)$. In this case, t_1 can only be in the pseudo-configuration form. Lemma 5 implies that t_2 is in the pseudo-configuration form too. As the formula $\phi_{\text{loop}}(*)$ is satisfied by S Lemma 4 leads to contradiction.

(C) $t_1 = \bot_g(s_1)$. In this case, t_1 must be in the pseudo-run form. At the same time, we have $t_1 \neq t_2$ by Lemma 3. If the first rule from the group [1] in Def. 10 is applied in the reduction step $t_1 \to^p t_2$ then t_2 is in the $\bot^1_g$-pseudo-run form. The remaining rules in the group cannot be applied to t_1. The rules in the group [2] cannot be applied either, as there is no occurrence of g in t_1. The rules in [3] cannot be applied as $t_1 \neq t_2$.

In case a rule from [4] is applied, then it cannot be applied at the root address of t_1. This means that $t_2 = \bot_g(s_2)$ and $s_1 \to^p s_2$ holds. Thus, Lemma 5 implies that s_2 is in the pseudo-configuration form so t_2 is in the weak pseudo-run form which leads to contradiction by Lemma 6 with the satisfiability of $\phi_{\text{loop}}(*)$.

The rules from the remaining group [5] cannot be applied as t_1 does not contain $\bot^i_g$ for $i = 1, 2, 3$.

(D) $t_1 = \bot^1_g(s_1)$. In this case, t_1 must be in the weak $\bot^1_g$-pseudo-run form. If $t_1 = t_2$ then the claim of the lemma automatically holds. We assume in the rest of the case that $t_1 \neq t_2$.

The first rule from the group [1] in Def. 10 cannot be applied as the left-hand side of the rule cannot match t_1. The second rule is impossible as $t_1 \neq t_2$. The third rule cannot be applied as then t_2 would be in the weak pseudo-run form and this would lead to contradiction by Lemma 6. The rules in group [2] cannot be applied to t_1 either, as there is no occurrence of g in t_1. The rules in [3] cannot be applied as $t_1 \neq t_2$.

In case a rule from [4] is applied, it cannot be applied at the root address of t_1. This means that $t_2 = \bot^1_g(s_2)$ and $s_1 \to^p s_2$ holds. Thus, Lemma 5 implies that s_2 is in the pseudo-configuration form. This immediately gives that t_2 is in the weak $\bot^1$-pseudo-run form.

In case the rule $\bot^1_g(x) \to \bot^2_g(x)$ from [5] is used to obtain t_2, we have two possibilities: either all the reductions in $t_2 \to^p t_{21} \to^p t_{22} \to^p t_2$ are not at the root address or some reduction at root address is used there. In the first case we obtain a sequence of the pseudo-configuration forms that satisfy the formula (4) which is impossible by Lemma 4. In the second case, we cannot use the rule $\bot^2_g(x) \to \bot^2_g(x)$ to rewrite $t_2 \to^p t_{21}$ as $t_2 \neq t_{21}$. Thus the only remaining possibility is to use $\bot^2_g(x) \to \bot^3_g(x)$, but then we have to obtain $\bot^2_g$ back at the end of the loop in t_2. This is possible only when $\bot^3_g(x) \to \bot^1_g(x)$ is used as this is the only way to dispose of $\bot^3_g$. In this way we used up all 3 reduction steps in the loop $t_2 \to^p t_{21} \to^p t_{22} \to^p t_2$. However, now we have also $t_2 \to^p t_{22}$ with the rule $\bot^2_g(x) \to \bot^1_g(x)$ so the formula $\phi_{\text{loop}}(*)$ cannot be satisfied in this case.

(E) $t_1 = \bot^i_g(s_1)$ for $i = 2, 3$. This case is impossible as terms of this form are not in the x_*-form.

(F) $t_1 = g(s_1, s_2)$. In this case t_1 and s_1 are both in the pseudo-run form or weak $\bot^1_g$-pseudo-run form while s_2 is in the pseudo-configuration form. If

$t_1 = t_2$ then the claim holds either because t_2 is in the required form or because the case can be discared by contradiction with Lemma 6. Thus, we assume now that $t_1 \neq t_2$.

If the rewriting $t_1 \to^p t_2$ is in the root then it can hold only due to the use of a rule in the group [2] (other rules have left-hand side that either does not match or results in $t_1 = t_2$). This means that either t_2 is in the weak $\bot^1_g$-pseudo-run form, which matches with our claim, or in the weak pseudo-run form. This last situation is, however, impossible by Lemma 6.

If the rewriting $t_1 \to^p t_2$ is not in the root address then $t_2 = g(s'_1, s'_2)$ where $s_i = s'_i$ or $s_i \to^p s'_i$ for $i = 1, 2$ and at least one of $s_i \to^p s'_i$ holds. The term s'_2 is in the pseudo-configuration form either as it is equal to s_2 or by Lemma 5. We may also assume that $s_1 \neq s'_1$ as otherwise we immediately obtain our claim that t_2 is in the weak $\bot^1_g$-pseudo-run form by Lemma 5. As t_1 is in the x_*-form it may be either in the pseudo-run form or in the weak $\bot^1_g$-pseudo-run form.

In case t_1 is in the pseudo-run form Lemma 5 implies that t_2 is either in the pseudo-run form, but this is impossible by Lemma 6, or in the weak $\bot^1_g$-pseudo-run form, which fulfils our claim.

In case t_1 is in the weak $\bot^1_g$-pseudo-run form the only rule which changes the form is the rule $\bot^1_g(x) \to \bot^2_g(x)$ (our claim is fulfiled when the form is not changed). The only reversible step in the reduction $t_2 \to^p t_{21}$ can be done according to the rules in [4]. Then the next step may be done either by the rules in [4], or by the rules in [5]. In the first case, t_{22} contains $\bot^2_g(x)$ and the only possible rewriting is again only by [4] and as this is reversible it leads to contradiction. In the second case, we have to perform at least two rewrites to get $\bot^2_g(x)$ back which is impossible to accomplish in the one rewriting step $t_{22} \to^p t_2$. This concludes this case so that the reduction $t_1 \to^p t_2$ with the use of the rule $\bot^1_g(x) \to \bot^2_g(x)$ is impossible. □