Technical Report

An Extension of Pushdown System and Its Model Checking Method

Naoya Nitta and
Hiroyuki Seki

{naoya-n,seki}@is.aist-nara.ac.jp

June 11, 2003

Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

In this paper, we present a class of infinite transition systems which is an extension of pushdown systems (PDS), and show that LTL (linear temporal logic) model checking for the class is decidable. Since the class is defined as a subclass of term rewriting systems, pushdown stack of PDS is naturally extended to tree structure. By this extension, we can model recursive programs with exception handling.

1 Introduction

Model checking [2] is a well-known technique which automatically verifies whether a system satisfies a given specification. Most of existing model checking methods and tools assume that a system to be verified has finite state space. This is a serious restriction when we apply model checking to software verification since a program is usually modeled as a system with infinite state space. There are two approaches to resolving the problem. One is that if a system to be verified has infinite state space, then the system is transformed into an abstract system with finite state space. This transformation is called abstraction and various abstraction methods have been proposed. Among them are predicate abstraction [12] and Bandera which uses program slicing and abstract interpretation [4]. However, the abstract system does not always retain the desirable property which the original system has, in which case the verification fails.

Another approach is to introduce a new subclass of transition systems which is wider than finite state systems and investigate model checking method for the subclass. Pushdown system is such a subclass that is wider than finite state systems and yet has decidable properties on model checking. A pushdown system (abbreviated as PDS) is an autonomous transition system with a pushdown stack as well as a finite control. A PDS can model a system which has well-nested structure such as a program involving recursive procedure calls. Recently, efficient algorithms of LTL and CTL* model checking for PDS have been proposed in [5, 6] (also see related works). The transition relation of a PDS is defined by transition rules which rewrite the finite control and a prefix of the string in the pushdown stack. Thus, if we model a program as a PDS, we are forced to define the behavior of the program by transition rules on strings.

In this paper, we focus on term rewriting system (abbreviated as TRS), which is one of the well-known general computation models, and define the model checking problem for TRS. For simplicity, we consider the rewrite relation induced by the rewriting only at the root position of a term (root rewriting). Since a transition in a PDS changes the finite control and a prefix of the strings in the stack, PDS can be regarded as a TRS with root rewriting. Next, a new subclass of TRS, called generalized-growing TRS (GG-TRS) is defined. GG-TRS properly includes growing TRS [18] of Nagaya and Toyama. We present a necessary and sufficient condition for a left-linear(LL-)GG-TRS $\mathcal{R}$ to have an infinite rewrite sequence which visits terms in a given set infinitely often. Based on this condition, we then present a condition for $\mathcal{R}$ to satisfy a given LTL formula ϕ. The latter condition is decidable if $\mathcal{R}$ has a property called pre-(or post-)recognizability preserving property. Lastly, we introduce a subclass of TRS called LL-SPO-TRS and show that every TRS in this subclass has pre-recognizability preserving property. Every PDS belongs to both of GG-TRS and LL-SPO-TRS. Furthermore, we show that a program with recursive procedure and exception handling can be naturally modeled as a TRS in both GG-TRS and LL-SPO-TRS, which is not strongly bisimilar to any PDS. In this sense, the decidability results on LTL model checking in this paper is an extension of the results in [5, 6].

Related Works The model checking problem for PDS and the modal μ-calculus is studied in [24]. For LTL and CTL*, efficient model checking algorithms for PDS are proposed in [5, 6]. Major applications of model checking for PDS are static analysis of programs and security verification. For the former, Esparza et al. [6] discuss an application of model checking for PDS to dataflow analysis of recursive programs. Some results obtained by using their verification tool including the termination analysis of a quicksort program are also reported in [7]. The first work which applies model checking of a pushdown-type system to security verification is Jensen et al.'s study [14]. In that paper, they formally define a verification problem for a program with access control which generalizes JDK1.2 stack inspection [11]. However, their approach has severe restrictions, e.g, a mutual recursion is prohibited. Nitta et al. [19, 20] improve the result of [14] by using indexed grammar in formal language theory and show that a verification problem is decidable for an arbitrary program with stack inspection. The problem for a program which contains *no* stack inspection in [14, 19] exactly corresponds to model checking of a safety property ($AG\psi$) with regular valuation [6] for a PDS. In [20], a subclass of programs which exactly represents programs with JDK1.2 stack inspection is proposed and it is shown that verification of the safety property can be performed in polynomial time of the program size in the subclass. They also present verification results by using a verification tool. In [6], it is shown that LTL model checking is decidable for an arbitrary programs with stack inspection. Jha and Reps show that name reduction in SPKI [22] can be represented as a PDS, and prove the decidability of a number of security problems by reductions to decidability properties of model checking for PDS [15]. Among other infinite state systems for which model checking has been studied are process rewrite system (PRS) [17] and ground TRS [16]. PRS includes PDS and Petri Net as its subclasses. However, LTL model checking is undecidable for both of PRS and ground TRS.

2 Preliminaries

2.1 Term Rewriting System

We use the usual notions for terms, substitutions, etc (see [1] for details). Let N denote the set of natural numbers. Let $\mathcal{F}$ be a *signature* and $\mathcal{V}$ be an enumerable set of *variables*. An element in $\mathcal{F}$ is called a *function symbol* and the *arity* of $f \in \mathcal{F}$ is denoted by $arity(f)$. A function symbol c with $arity(c) = 0$ is called a *constant*. The set of *terms* generated by $\mathcal{F}$ and $\mathcal{V}$ is defined in the usual way and denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The set of variables occurring in t is denoted by $\mathcal{V}ar(t)$. A term t is *ground* if $\mathcal{V}ar(t) = \emptyset$. The set of all ground terms is denoted by $\mathcal{T}(\mathcal{F})$. A term is *linear* if no variable occurs more than once in the term. A *substitution* θ is a mapping from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and written as $\theta = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where t_i with $1 \leq i \leq n$ is a term which substitutes for the variable x_i. The term obtained by applying a substitution θ to a term t is written as $t\theta$. We call $t\theta$ an *instance* of t. A *position* in a term t is defined as a sequence of positive integers as usual, and the root position is the empty sequence denoted by λ. The depth of a position $p \in (N - \{0\})^*$, written as $|p|$, is the length of p (e.g. $|132| = 3$). Let $\preceq_{pref}$ denote the prefix relation on positions, defined as usual. The set of all positions in a term t is denoted by $Pos(t)$. Also let us define $Pos_{=n}(t) = \{p \in Pos(t) \mid |p| = n\}$ and $Pos_{\geq n}(t) = \{p \in Pos(t) \mid |p| \geq n\}$. A subterm of t at a position $p \in Pos(t)$ is denoted by $t_{|p}$. $Pos(t, s)$ is the set $\{p \mid t_{|p} = s\}$. If $t_{|p} = f(\cdots)$, then we write $lab(t, p) = f$. If a term t is obtained from a term t' by replacing the subterms of t' at positions $p_1, \ldots, p_m$ ($p_i \in Pos(t'), p_i$ and p_j are disjoint if $i \neq j$) with terms $t_1, \ldots, t_m$, respectively, then we write $t = t'[p_i \leftarrow t_i \mid 1 \leq i \leq m]$. The depth of a term t is $max\{|p| \mid p \in Pos(t)\}$. For terms s, t, let $mgu(s, t)$ denote the most general unifier of s and t if it is defined. Otherwise, let $mgu(s, t) = \perp$.

A *rewrite rule* over a signature $\mathcal{F}$ is an ordered pair of terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, written as $l \rightarrow r$. A *term rewriting system* (*TRS*) over $\mathcal{F}$ is a finite set of rewrite rules over $\mathcal{F}$. For terms t, t' and a TRS $\mathcal{R}$, we write $t \rightarrow_{\mathcal{R}} t'$ if there exists a position $p \in \mathcal{P}os(t)$, a substitution θ and a rewrite rule $l \rightarrow r \in \mathcal{R}$ such that $t/p = l\theta$ and $t' = t[p \leftarrow r\theta]$. Define $\rightarrow^*_{\mathcal{R}}$ to be the reflexive and transitive closure of $\rightarrow_{\mathcal{R}}$. Sometimes $t \rightarrow^*_{\mathcal{R}} t'$ is called a rewrite sequence. Also the transitive closure of $\rightarrow_{\mathcal{R}}$ is denoted by $\rightarrow^+_{\mathcal{R}}$. The subscript $\mathcal{R}$ of $\rightarrow_{\mathcal{R}}$ is omitted if $\mathcal{R}$ is clear from the context. A *redex* (*in* $\mathcal{R}$) is an instance of l for some $l \to r \in \mathcal{R}$. A *normal form* (in $\mathcal{R}$) is a term which has no redex as its subterm. Let $\mathrm{NF}_{\mathcal{R}}$ denote the set of all ground normal forms in $\mathcal{R}$. A rewrite rule $l \to r$ is *left-linear*(resp. *right-linear*) if l is linear (resp. r is linear). A rewrite rule is *linear* if it is left-linear and right-linear. A TRS $\mathcal{R}$ is *left-linear* (resp. *right-linear, linear*) if every rule in $\mathcal{R}$ is left-linear (resp. right-linear, linear).

2.2 Tree Automata and Recognizability

A *tree automaton*(*TA*) [8] is defined by a 4-tuple $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \Delta, \mathcal{Q}^{\mathit{final}})$ where $\mathcal{F}$ is a signature, $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}^{\mathit{final}} \subseteq \mathcal{Q}$ is a set of final states, and Δ is a finite set of transition rules of the form $f(q_1, \ldots, q_n) \to q$ where $f \in \mathcal{F}$, $\mathit{arity}(f) = n$, and $q_1, \ldots, q_n, q \in \mathcal{Q}$ or of the form $q' \to q$ where $q, q' \in \mathcal{Q}$. Consider the set of ground terms $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ where we define $\mathit{arity}(q) = 0$ for $q \in \mathcal{Q}$. A *transition* of a TA can be regarded as a rewrite relation on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ by regarding transition rules in Δ as rewrite rules over $\mathcal{F} \cup \mathcal{Q}$. For terms t and t' in $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, we write $t \vdash_{\mathcal{A}} t'$ if and only if $t \to_{\Delta} t'$. The reflexive and transitive closure and the transitive closure of $\vdash_{\mathcal{A}}$ is denoted by $\vdash_{\mathcal{A}}^{*}$ and $\vdash_{\mathcal{A}}^{+}$, respectively. For a TA $\mathcal{A}$ and $t \in \mathcal{T}(\mathcal{F})$, if $t \vdash_{\mathcal{A}}^{*} q_f$ for a final state $q_f \in \mathcal{Q}^{\mathit{final}}$, then we say t is *accepted* by $\mathcal{A}$. The set of all ground terms in $\mathcal{T}(\mathcal{F})$ accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$ and we say that $\mathcal{A}$ recognizes $\mathcal{L}(\mathcal{A})$. A subset $L \subseteq \mathcal{T}(\mathcal{F})$ of ground terms is called a tree language. A tree language L is *recognizable* if there is a TA $\mathcal{A}$ such that $L = \mathcal{L}(\mathcal{A})$.

Lemma 2.1 *[8] The class of recognizable tree languages is effectively closed under union, intersection and complementation. For given recognizable tree languages L_1 and L_2, the inclusion problem ($L_1 \subseteq L_2$?) is decidable. (Thus, membership ($t \in L$?), emptiness ($L = \emptyset$?), and equivalence ($L_1 = L_2$?) are also decidable.)* $\square$

For a TRS $\mathcal{R}$ and a tree language L, let $\mathit{post}^{*}_{\mathcal{R}}(L) = \{t \mid \exists s \in L \text{ s.t. } s \to^{*}_{\mathcal{R}} t\}$ and $\mathit{pre}^{*}_{\mathcal{R}}(L) = \{t \mid \exists s \in L \text{ s.t. } t \to^{*}_{\mathcal{R}} s\}$. A TRS $\mathcal{R}$ is said to *effectively preserve post-recognizability* (abbreviated as post-PR) if, for any TA $\mathcal{A}$, $\mathit{post}^{*}_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$ is also recognizable and we can effectively construct a TA which accepts $\mathit{post}^{*}_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$. We define pre-PR in a similar way. For a TRS $\mathcal{R}$, let $\mathcal{R}^{-1} = \{r \to l \mid l \to r \in \mathcal{R}\}$.

By definition, $post^*_{\mathcal{R}^{-1}}(L) = pre^*_{\mathcal{R}}(L)$. Thus, a TRS $\mathcal{R}$ is pre-PR if and only if $\mathcal{R}^{-1}$ is post-PR. Due to the properties of recognizable tree languages mentioned in Lemma 2.1, some important problems, e.g., reachability, joinability and local confluence are decidable for post-PR TRS [10, 13]. However, whether a given TRS is pre-PR (post-PR) is undecidable [9], and decidable subclasses of pre-PR or post-PR TRS have been proposed, some of which are listed with inclusion relation:

> RL-SM(semi-monadic)-TRS[3] $\subset$ RL-GSM(generalized semi-monadic)-TRS [13] $\subset$ RL-FPO(finitely path overlapping)-TRS[23]

where RL stands for 'right-linear.' As a decidable subclass of pre-PR TRS, left-linear growing TRS (LL-G-TRS) [18] is known. A TRS $\mathcal{R}$ is a G-TRS if for every rule $l \to r$ in $\mathcal{R}$, every variable in $\mathcal{V}ar(l) \cap \mathcal{V}ar(r)$ appears at depth 0 or 1 in l. Hence, a shallow TRS is always a G-TRS. Note that $\mathcal{R}$ is an SM-TRS if and only if $\mathcal{R}^{-1}$ is a G-TRS and the left-hand side of any rule in $\mathcal{R}$ is not a constant.

2.3 Transition Systems and Linear Temporal Logic

A *transition system* is a 3-tuple $\mathcal{S} = (S, \to, s_0)$, where S is a (possibly infinite) set of states, $\to \subseteq S \times S$ is a *transition relation* and $s_0 \in S$ is an *initial state*. The transitive closure of $\to$ and the reflexive and transitive closure of $\to$ are written by $\to^+$ and $\to^*$, respectively. A *run* of $\mathcal{S}$ is an infinite sequence of states $\sigma = s_1 s_2 \ldots$ such that $s_i \to s_{i+1}$ for each $i \geq 1$. Let $At = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ be a set of atomic propositions. The syntax of linear temporal logic (LTL) formula ϕ is defined by

$$\phi ::= tt \mid \alpha_i \mid \neg\phi \mid \phi_1 \wedge \phi_1 \mid \mathcal{X}\phi \mid \phi_1 \mathcal{U} \phi_2$$

($1 \leq i \leq k$ and ϕ_1, ϕ_2 are LTL formulas). For a transition system $\mathcal{S} = (S, \to, s_0)$, a *valuation* of $\mathcal{S}$ is a function $\nu : At \to 2^S$. The *validity* of an LTL formula ϕ for a run $\sigma = s_1 s_2 \ldots$ w.r.t. a valuation ν is denoted by $\sigma \models^\nu \phi$, and defined in the standard way [2]. We say ϕ is *valid* at s w.r.t. ν, denoted as $s \models^\nu \phi$, if and only if $\sigma \models^\nu \phi$ for each run σ starting in s.

2.4 Model Checking for TRS

Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$ and a term $t_0 \in \mathcal{T}(\mathcal{F})$, we can define a transition system $\mathcal{S}_\mathcal{R} = (\mathcal{T}(\mathcal{F}), \to_{\mathcal{S}_\mathcal{R}}, t_0)$ where $\to_{\mathcal{S}_\mathcal{R}} = \to_\mathcal{R} \cup \{(t,t) \mid t \in \mathrm{NF}_\mathcal{R}\}$. Note that the reflexive relation $\{(t,t) \mid t \in \mathrm{NF}_\mathcal{R}\}$ is needed to make the transition relation $\to_{\mathcal{S}_\mathcal{R}}$ total. The validity of LTL formula ϕ at t_0 in $\mathcal{S}_\mathcal{R}$ w.r.t. $\nu : At \to 2^{\mathcal{T}(\mathcal{F})}$ is denoted as $\mathcal{R}, t_0 \models^\nu \phi$. From an LTL formula ϕ, we can construct a Büchi automaton which recognizes the set of models of $\neg\phi$. Therefore, in the following, we often assume that we are given a Büchi automaton instead of an LTL formula. In a similar way to the model checking method in [6], below, we give a definition of *Büchi TRS* which synchronizes a transition system $\mathcal{S}_\mathcal{R}$ given by a TRS $\mathcal{R}$ with a Büchi automaton $\mathcal{B}$. First, to make the definition constructive, we make a few observations. To synchronize $\mathcal{S}_\mathcal{R}$ with $\mathcal{B}$, we must construct a Büchi TRS so that the redex can keep track of the information on the current state of $\mathcal{B}$ and the valuation of the current term of $\mathcal{S}_\mathcal{R}$. However, if we are given a TRS $\mathcal{R}$ of which an arbitrary redex can be rewritten, transmitting the above information to the next redex in the Büchi TRS becomes difficult. For this reason, we consider *root rewriting*, which restricts rewriting positions to the root position. Formally, root rewriting is defined as follows.

Definition 2.1 (**Root Rewriting**) For terms t, t' and a TRS $\mathcal{R}$, we say $t \to_\mathcal{R} t'$ is *root rewriting*, if there exist a substitution θ and a rewrite rule $l \to r \in \mathcal{R}$ such that $t = l\theta$ and $t' = r\theta$. $\square$

If we consider root rewriting, it is not difficult to see that there effectively exists a TRS of which the rewrite relation exactly corresponds to $\to_{\mathcal{S}_\mathcal{R}}$. Let $\{\Lambda_1, \ldots, \Lambda_m\}$ be a set of terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\mathrm{NF}_\mathcal{R} = \bigcup_{1 \le i \le m}\{\Lambda_i\theta \mid \theta : \mathcal{V} \to \mathcal{T}(\mathcal{F})$ is a substitution$\}$, and $mgu(\Lambda_i, \Lambda_j) = \bot (1 \le i < j \le m)$. Also, let $\widetilde{\mathcal{R}} = \mathcal{R} \cup \{\Lambda_i \to \Lambda_i \mid 1 \le i \le m\}$. Then, $t \in \mathrm{NF}_\mathcal{R}$ if and only if there exists a unique Λ_i such that $t \to_{\widetilde{\mathcal{R}}} t \to_{\widetilde{\mathcal{R}}} \cdots$ where $\Lambda_i \to \Lambda_i$ is applied in each rewrite step. Hence, we know $\to_{\mathcal{S}_\mathcal{R}} = \to_{\widetilde{\mathcal{R}}}$, i.e., the transition relation $\to_{\mathcal{S}_\mathcal{R}}$ of $\mathcal{S}_\mathcal{R}$ can be induced by TRS $\widetilde{\mathcal{R}}$. Next, we extend the definitions of valuations of PDS [5, 6].

Definition 2.2 (**Simple Valuation**)
Let $\mu : At \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a function such that for each $\alpha \in At$ and $l \to r \in \widetilde{\mathcal{R}}$,

$mgu(l, \mu(\alpha)) = l$ or $=\perp$. The *simple valuation* $\nu : At \to 2^{\mathcal{T}(\mathcal{F})}$ given by μ is defined as $\nu(\alpha) = \{\mu(\alpha)\theta \mid \theta \text{ is a substitution}\}$. $\square$

In the definition, $\mu(\alpha)$ specifies a pattern of terms for which proposition α is true. For example, if $\mu(\alpha_1) = f(x, g(y))$ then $\mathcal{R}, t \models^{\nu} \alpha_1$ if and only if t is an instance of $f(x, g(y))$. The restriction that $mgu(l, \mu(\alpha)) = l$ or $=\perp$ guarantees that for a rewrite rule $l \to r$, whether $\mathcal{R}, l\theta \models^{\nu} \alpha$ is determined independent of a substitution θ.

Definition 2.3 (**Regular Valuation**)
For each atomic proposition $\alpha \in At$, a TA $\mathcal{A}_\alpha$ is given. The *regular valuation* $\nu : At \to 2^{\mathcal{T}(\mathcal{F})}$ given by $\langle \mathcal{A}_\alpha \rangle_{\alpha \in At}$ is defined as $\nu(\alpha) = \mathcal{L}(\mathcal{A}_\alpha)$. $\square$

Definition 2.3 says that $\mathcal{R}, t \models^{\nu} \alpha$ if and only if t is accepted by $\mathcal{A}_\alpha$. This is a natural extension of regular valuation ν of PDS, where a configuration $\langle q, w \rangle$ is a pair of a control location q and a sequence w of stack symbols and $\langle q, w \rangle \models^{\nu} \alpha$ if and only if the sequence qw is accepted by a finite state automaton $\mathcal{A}_\alpha$ given for α.

Definition 2.4 (**Büchi TRS**) Let At be a set of atomic propositions, $\mathcal{R}$ be a TRS, $\mathcal{B} = (\mathcal{Q}_\mathcal{B}, \Sigma_\mathcal{B}, \Delta_\mathcal{B}, q_{0\mathcal{B}}, \mathcal{Q}_\mathcal{B}^{acc})$ ($\Sigma_\mathcal{B} = 2^{At}$, $\mathcal{Q}_\mathcal{B} \cap \mathcal{F} = \emptyset$) be a Büchi automaton, and ν be the simple valuation given by $\mu : At \to \mathcal{T}(\mathcal{F}, \mathcal{V})$. For $\mathcal{R}$, $\mathcal{B}$ and ν, we define Büchi TRS $\mathcal{BR}^{\nu}$ as follows: The signature of $\mathcal{BR}^{\nu}$ is $\mathcal{F}_{\mathcal{BR}^{\nu}} = \mathcal{Q}_\mathcal{B} \cup \mathcal{F}$(for any $q \in \mathcal{Q}_\mathcal{B}$, $arity(q) = 1$), and $\mathcal{BR}^{\nu}$ is the minimum set of rules satisfying:

$$\begin{aligned} & q \xrightarrow{a} q' \in \Delta_\mathcal{B}, l \to r \in \widetilde{\mathcal{R}}, \text{and } \ a \subseteq \{\alpha \in At \mid mgu(l, \mu(\alpha)) = l\} \\ & \Rightarrow q(l) \to q'(r) \in \mathcal{BR}^{\nu}. \end{aligned}$$

$\square$

If ν is regular, then we can reduce a model checking problem w.r.t. ν to a model checking problem w.r.t. a simple valuation in a similar way to [6].

Lemma 2.2 *Let $\mathcal{R}$ be a TRS, $t_0 \in \mathcal{T}(\mathcal{F})$ be an initial state, ϕ be an LTL formula, $\mathcal{B} = (\mathcal{Q}_\mathcal{B}, \Sigma_\mathcal{B}, \Delta_\mathcal{B}, q_{0\mathcal{B}}, \mathcal{Q}_\mathcal{B}^{acc})$ ($\Sigma_\mathcal{B} = 2^{At}$, $\mathcal{Q}_\mathcal{B} \cap \mathcal{F} = \emptyset$) be a Büchi automaton which represents $\neg\phi$, and ν be the simple valuation given by $\mu : At \to \mathcal{T}(\mathcal{F}, \mathcal{V})$. Also, let $\mathcal{T}_{\mathrm{acc}} = \{q_{\mathrm{a}}(t) \mid q_{\mathrm{a}} \in \mathcal{Q}_\mathcal{B}^{acc}, t \in \mathcal{T}(\mathcal{F})\}$. $\mathcal{R}, t_0 \not\models^{\nu} \phi$ if and only if there exists an infinite root rewrite sequence of Büchi TRS $\mathcal{BR}^{\nu}$ starting in $q_{0\mathcal{B}}(t_0)$ and visiting $\mathcal{T}_{\mathrm{acc}}$ infinitely often.* $\square$

3 Generalized-Growing TRS and Its Model Checking

The restriction of root rewriting (Definition 2.1) on TRS $\mathcal{R}$ is insufficient to make the model checking problem for $\mathcal{R}$ decidable, because root rewriting TRSs are still Turing powerful. In fact, we can define an automaton with two pushdown stacks (which is Turing powerful) as a left-linear root rewriting TRS by encoding a state of the finite control as a root symbol q with arity 2 and each of the two stacks as each argument of q. The reason why root rewriting TRSs are Turing powerful is unrestricted information flow between different arguments of a function symbol such as q above. We introduce a subclass of TRS, called LL-GG-TRS, in which the information of (function symbol in) an argument is never shifted to another argument, and show that if an LL-GG-TRS $\mathcal{R}$ is post-PR (or pre-PR), then LTL model checking for $\mathcal{R}$ is decidable. For positions p_1, p_2, we define the *least common ancestor* $p_1 \sqcup p_2$ as the longest common prefix of p_1 and p_2.

Definition 3.1 (**Left-Linear Generalized-Growing TRS (LL-GG-TRS)**) A left-linear rule $l \to r$ is *generalized-growing*, if every two different variables $x, y \in \mathcal{V}ar(l) \cap \mathcal{V}ar(r)$ satisfy the following condition: For the positions o_l^x, o_l^y of x, y in l and for each positions $o_r^x \in Pos(r, x), o_r^y \in Pos(r, y)$ of x, y in r,

$$|o_l^x| - |o_l^x \sqcup o_l^y| \ \leq \ |o_r^x| - |o_r^x \sqcup o_r^y|, \text{ and } |o_l^y| - |o_l^x \sqcup o_l^y| \ \leq \ |o_r^y| - |o_r^x \sqcup o_r^y|.$$

$\mathcal{R}$ is left-linear generalized-growing (LL-GG), if every rule in TRS $\mathcal{R}$ is left-linear and generalized-growing. □

Obviously, an LL-G-TRS (see section 2.2) is always an LL-GG-TRS.

Example 3.1 Consider $\mathcal{R}_1 = \{ \ f(g(x, y)) \to f(h(y), x) \ \}$. The position of x is 11 in l and 2 in r, and the position of y is 12 in l and 11 in r. Since $11 \sqcup 12 = 1$ and $2 \sqcup 11 = \lambda$, $\mathcal{R}_1$ is an LL-GG-TRS, but $\mathcal{R}_1$ is not an LL-G-TRS because variables x and y occur at depth 2 in l. On the other hand, $\mathcal{R}_1^{-1} = \{ \ f(h(y), x) \to f(g(x, y)) \ \}$ is not an LL-GG-TRS, since the difference of the depth of positions in l between y and the least common ancestor of x and y is larger than that in r. □

```
    void main() {
L0:     switch (random_integer) {
        case 0:
L1:        main();
           break;
        case 1:
L2:        try {
L3:           main();
           } catch (e) {
L4:           nop;
           }
           break;
        case 2:
L5:        throw e;
        }
L6:     return;
    }
```

$\Rightarrow$

$$
\begin{array}{ll}
\mathcal{R}_2 = \{ & \\
\left.\begin{array}{l}
L_0(x,y) \to L_1(x,y), \\
L_0(x,y) \to L_2(x,y), \\
L_0(x,y) \to L_5(x,y), \\
L_0(x,y) \to L_6(x,y), \\
L_4(x,y) \to L_6(x,y),
\end{array}\right\} & \textbf{: seq} \\
L_6(x,y) \to x, & \textbf{: ret} \\
L_5(x,y) \to y, & \textbf{: throw} \\
\left.\begin{array}{l}
L_1(x,y) \to L_0(L_6(x,y),y), \\
L_3(x,y) \to L_0(L_6(x,y),y),
\end{array}\right\} & \textbf{: call} \\
L_2(x,y) \to L_3(x, L_4(x,y)) & \textbf{: try-catch} \\
\} &
\end{array}
$$

Figure 1: A sample program with exception handling

Example 3.2 (**Recursive Program with Exception Handling**)
It is well-known that a program with recursive procedure can be naturally modeled as a PDS, and further in [21], a PDS model of Java-like programs including exception handling was proposed. In this model, the exception handling mechanism is implemented by adding extra control states and rules which represent low-level operations embedded in the language processing system. On the other hand, in this example, we present an LL-GG-TRS model of recursive programs, which is closer to the behavioral semantics incorporated with exception handling in the source code level. For example, a Java-like program in the left half of Fig.1 can be directly modeled as an LL-GG-TRS $\mathcal{R}_2$ shown in the right half. Note that the class LL-GG-TRS is propery wider than the class of PDSs w.r.t. strong bisimulation equivalence, and $\mathcal{R}_2$ is an example of LL-GG-TRS which has no strongly bisimilar PDS. In a Java program, try-catch-throw statements are used for specifying exception handling. By the execution of a *throw* statement, an exception is propagated in the program. If an exception occurs within a *try* block, then the control immediately moves to the *catch* statement coupled with the try

statement (with unwinding the control stack). From a program *Prog* including try-catch-throw statements, we can construct an LL-GG-TRS $\mathcal{R}$ as follows. In $\mathcal{R}$, a program location in *Prog* is represented by a function symbol and a state of *Prog* is expressed by a term. Every term t has the form of $f(t_1, t_2)$ where f denotes the current program location of *Prog*, t_1 denotes the next state of t if a return statement is executed at t, and t_2 denotes the next state of t if an exception occurs at t. A constant symbol $\Box$ denotes the stack bottom. Every unit executions of *Prog* are divided into five types, **seq**, **call**, **ret**, **try-catch** and **throw**. Each rule $l \to r \in \mathcal{R}$ represents a unit execution of *Prog* and according to the type of the execution, $l \to r$ has one of the following forms:

$$\begin{aligned}
\textbf{seq:}\quad & current(x,y) \to succ(x,y),\\
\textbf{call:}\quad & caller(x,y) \to callee(succ(x,y),y),\\
\textbf{ret:}\quad & ret(x,y) \to x,\\
\textbf{try-catch:}\quad & try(x,y) \to succ(x, catch(x,y)),\\
\textbf{throw:}\quad & throw(x,y) \to y.
\end{aligned}$$

A **seq** rule represents a sequential execution in a method in *Prog* where *succ* denotes an immediate successor of *current*. A **call** rule represents a method invocation in *Prog* where *calee* denotes the entry point of the method invoked by *caller* and *succ* denotes an immediate successor of *caller*. A **try-catch** rule represents the behavior of a try-catch block and *try*, *succ* and *catch* denote the program location of the try statement, the entry point of the try block and the entry point of the catch block, respectively. A **ret** rule represents a return statement and a **throw** rule represents a throw statement. It is interesting to recognize a symmetry between (**call, ret**) rules and (**try-catch, throw**) rules. Recall the program in Fig.1. Since the statement at L2 is try, the entry point of the try block is L3, and the entry point of the catch block is L4, $L_2(x,y) \to L_3(x, L_4(x,y)) \in \mathcal{R}_2$. $\Box$

In the following, we only consider root rewrite sequences consisting of ground terms. The first lemma for LL-GG-TRS states that for any root rewrite sequence σ if there exists a position o_0 in the first term t_0 of σ such that the depth of (a residual of) o_0 is never shortened in σ, then for every 'sufficiently deep' position p_0 in t_0, every residual of p_0 never be contained in any redex. For a TRS $\mathcal{R}$, let $max_v(\mathcal{R})$ be the maximum depth of positions of variables in the left-hand sides of rules in $\mathcal{R}$, and $max_f(\mathcal{R})$ be the maximum depth of positions of function symbols in both sides of rules in $\mathcal{R}$. For a rewrite sequence $\sigma : t \to_{\mathcal{R}}^* t'$ and $p \in Pos(t)$, the set of *residuals* of p in σ, denoted as $Res(p, \sigma)$, is defined as follows. $Res(p, t \to_{\mathcal{R}}^0 t) = \{p\}$. Assume $t = l\theta \to_{\mathcal{R}} r\theta = t'$ for a rule $l \to r$ and a substitution θ.

$$Res(p, t \to_{\mathcal{R}} t') = \begin{cases} \{p'_1 p_2 \mid r_{|p'_1} = x\} & \text{if } p = p_1 p_2 \text{ and } l_{|p_1} = x \in Var(l), \\ \emptyset & \text{otherwise.} \end{cases}$$

For a rewrite sequence $t \to_{\mathcal{R}}^* t' \to_{\mathcal{R}} t''$, $Res(p, t \to_{\mathcal{R}}^* t' \to_{\mathcal{R}} t'') = \{p'' \mid p' \in Res(p, t \to_{\mathcal{R}}^* t') \text{ and } p'' \in Res(p', t' \to_{\mathcal{R}} t'')\}$. We abbreviate $Res(p, t \to_{\mathcal{R}}^* t')$ as $Res(p, t')$ if the sequence $t \to_{\mathcal{R}}^* t'$ is clear from the context.

Lemma 3.1 *Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. Also let $\sigma = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_{k-1} \to_{\mathcal{R}} t_k \to_{\mathcal{R}} t_{k+1}$ of $\mathcal{R}$ be a root rewrite sequence and $o_0 \in Pos(t_0)$ be a position. If there exists a position $o_i \in Res(o_0, t_i)$ such that $|o_0| \leq |o_i|$ for each $i (1 \leq i \leq k)$, then every position $p_0 \in Pos_{\geq c}(t_0)$ satisfies the following (a) and (b):*

(a) For an arbitrary $p_k \in Res(p_0, t_k)$,

$$|p_k| - |o_k \sqcup p_k| \geq |p_0| - |o_0 \sqcup p_0| \text{ and } |p_k| > max_f(\mathcal{R}).$$

(b) For an arbitrary $s \in \mathcal{T}(\mathcal{F})$,

$$t_0[p_0 \leftarrow s] \to_{\mathcal{R}}^* t_{k+1}[Res(p_0, t_{k+1}) \leftarrow s].$$

Proof Sketch. (a) By induction on the length k of σ (see Appendix for the detail). (b) By (a), each $p_i \in Res(p_0, t_i) (0 \leq i \leq k)$ satisfies $|p_i| > max_f(\mathcal{R})$. Hence, we can construct a rewrite sequence starting in $t_0[p_0 \leftarrow s]$, applying the rules in the same order as σ. $\square$

The next lemma states that for any infinite root rewrite sequence σ of an LL-GG-TRS and any term t_n in σ, one can find a term t_m after t_n such that every 'sufficiently deep' position in t_m does not affect the rewrite sequence after t_m.

Definition 3.2 (**Longest-living position**) Let $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots$ be a rewrite sequence and $o_0 \in Pos(t_0)$ be a position. The *lifetime* of o_0 (in t_0) is defined as k, if there exists k such that $Res(o_0, t_i) \neq \emptyset$ $(0 \leq i \leq k)$ and $Res(o_0, t_i) = \emptyset$ $(i > k)$. Otherwise ($Res(o_0, t_i) \neq \emptyset$ for any $i \geq 0$), the lifetime of o_0 is undefined. A position which has the maximum lifetime in t_0 is called the *longest-living* position, if the lifetime of every position in t_0 is defined. □

Lemma 3.2 *Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. If there exists an infinite root rewrite sequence $\sigma = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots$ of $\mathcal{R}$, then for any $n \geq 0$, there exists $m > n$ such that for every $p_m \in Pos_{\geq c}(t_m)$, $k > m$ and $s \in \mathcal{T}(\mathcal{F})$:*

$$t_m[p_m \leftarrow s] \to^*_{\mathcal{R}} t_k[Res(p_m, t_k) \leftarrow s].$$

Proof Sketch. Assume that there exists a position p_n in t_n of which the lifetime is undefined (the proof for the other case is included in Appendix). Let p_i be the deepest residual of p_n in $t_i (i > n)$, and m be the minimum $j (> n)$ such that $|p_j| \leq |p_i|$ for each $i (> j)$. Note that m is always defined since $t_n \to_{\mathcal{R}} t_{n+1} \to_{\mathcal{R}} \ldots$ is an infinite sequence. Also, $t_m \to_{\mathcal{R}} t_{m+1} \to_{\mathcal{R}} \ldots$ and p_m satisfy the hypothesis of Lemma 3.1. Hence, by Lemma 3.1(b), the lemma holds. □

Definition 3.3 (**Inclusion order $\sqsupseteq_a$**) The inclusion order $\sqsupseteq_a$ w.r.t. constant a is the least relation satisfying the following condition:

- For any term t, $t \sqsupseteq_a a$.
- If $t_1 \sqsupseteq_a t'_1, t_2 \sqsupseteq_a t'_2, \ldots t_n \sqsupseteq_a t'_n$, then $f(t_1, t_2, \ldots, t_n) \sqsupseteq_a f(t'_1, t'_2, \ldots, t'_n)$.

□

In the rest of this section, we assume a is a new constant which is not a member of $\mathcal{F}$. For a term $t \in \mathcal{T}(\mathcal{F} \cup \{a\})$, let $|t|_a$ denote $|Pos(t, a)|$. When a tuple of terms $\vec{\theta} = \langle \theta_1, \ldots, \theta_n \rangle \in \mathcal{T}^n(\mathcal{F} \cup \{a\})$ is given where $n = |t|_a$, let $t\vec{\theta}$ denote $t[p_i \leftarrow \theta_i \mid 1 \leq i \leq n]$ for $Pos(t, a) = \{p_1, \ldots, p_n\}$, by slightly abusing the notation. The following lemma states that every infinite root rewrite sequence of an LL-GG-TRS has a kind of cyclic property.

Lemma 3.3 *Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. For an infinite root rewrite sequence $\sigma = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots$ of $\mathcal{R}$, there exist a term $t_{\mathrm{R}} \in \mathcal{T}(\mathcal{F} \cup \{a\})(n = |t_{\mathrm{R}}|_a)$ of which the depth is c or less and tuples of terms $\vec{\theta} \in \mathcal{T}^n(\mathcal{F} \cup \{a\}), \vec{\theta'} \in \mathcal{T}^n(\mathcal{F})$ such that $t_{\mathrm{R}} \in pre^*(\{t_{\mathrm{R}}\vec{\theta}\})$ and $t_0 \in pre^*(\{t_{\mathrm{R}}\vec{\theta'}\})$ hold.*

Let $\mathcal{T}_G \subseteq \mathcal{T}(\mathcal{F} \cup \{a\})$ be a set of terms, which is downward-closed w.r.t $\sqsubseteq_a$. If terms in $\mathcal{T}_G$ appear infinitely often in σ, then $t_{\mathrm{R}} \in pre^*(\mathcal{T}_G \cap pre^+(\{t_{\mathrm{R}}\vec{\theta}\}))$ and $t_0 \in pre^*(\{t_{\mathrm{R}}\vec{\theta'}\})$ hold.

Proof. We define an infinite sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$ of infinite sequences and a function $f : N \to N$ as follows (Fig.2). The kth element of σ_i is denoted as $\sigma_i(k)$.

- $i = 0$: $\sigma_0 = \sigma$.
- $i > 0$: $f^i(0)$ is defined as m in Lemma 3.2 when infinite root rewrite sequence σ_{i-1} and $n = f^{i-1}(0)$ are given. $\sigma_i(k)$ is defined according to k as follows:
  - $k < f^i(0)$: $\sigma_i(k)$ is undefined.
  - $k = f^i(0)$:
  $$\sigma_i(k) = \sigma_{i-1}(k)[Pos_{=c}(\sigma_{i-1}(k)) \leftarrow a]. \tag{3.1}$$
  - $k > f^i(0)$: By the definition of $f^i(0)$, we can use Lemma 3.2 and obtain:
  $$\sigma_{i-1}(f^i(0))[Pos_{=c}(\sigma_{i-1}(f^i(0))) \leftarrow a] \quad \to^*_{\mathcal{R}} \sigma_{i-1}(k)[\mathcal{P}^{i,k} \leftarrow a], \tag{3.2}$$
  where:
  $\mathcal{P}^{i,k} = Res(Pos_{=c}(\sigma_{i-1}(f^i(0))), \sigma_{i-1}(k)) \subseteq Pos_{\geq(max_f(\mathcal{R})+1)}(\sigma_{i-1}(k))$.
  Now, let
  $$\sigma_i(k) = \sigma_{i-1}(k)[\mathcal{P}^{i,k} \leftarrow a], \tag{3.3}$$
  then (3.2) can be written as $\sigma_i(f^i(0)) \to^*_{\mathcal{R}} \sigma_i(k)$ by (3.1).

For the infinite sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$,

$$\sigma_0(k) \sqsupseteq_a \sigma_1(k) \sqsupseteq_a \sigma_2(k) \sqsupseteq_a \cdots \sqsupseteq_a \sigma_j(k) \ (f^j(0) \leq k < f^{j+1}(0)) \tag{3.4}$$

holds by (3.3). Now, we consider the infinite sequence $\sigma_1(f(0)), \sigma_2(f^2(0)), \ldots$ by picking up the 'diagonal' terms. Then, the depths of these terms are always c or less. By this fact, we can see that there exist an integer i and an infinite sequence $i < j_0 < j_1 < j_2 < \cdots$ of numbers such that for every $j_h (h \geq 0)$,

$$\sigma_i(f^i(0)) = \sigma_{j_h}(f^{j_h}(0)). \tag{3.5}$$

By (3.4) and (3.5), $\sigma_i(f^i(0)) \sqsubseteq_a \sigma_i(f^{j_0}(0))$. Hence, for $t_{\mathrm{R}} = \sigma_i(f^i(0))$, there exists $\vec{\theta} \in \mathcal{T}^n(\mathcal{F} \cup \{a\})$ such that $\sigma_i(f^{j_0}(0)) = t_{\mathrm{R}}\vec{\theta}$. Since $\sigma_i(f^i(0)) \to^*_{\mathcal{R}} \sigma_i(f^{j_0}(0))$, $t_{\mathrm{R}} \in pre^*(\{t_{\mathrm{R}}\vec{\theta}\})$ holds. Similarly, by (3.4), we can obtain $\sigma_i(f^i(0))(= t_{\mathrm{R}}) \sqsubseteq_a \sigma_0(f^i(0))(= t_{\mathrm{R}}\vec{\theta'})$ for some $\vec{\theta'} \in \mathcal{T}^n(\mathcal{F})$, and thus $\sigma_0(0)(= t_0) \in pre^*(\{t_{\mathrm{R}}\vec{\theta'}\})$ holds. By (3.1), the depth of t_{R} is c or less. Next, we consider the case that terms in $\mathcal{T}_G$ appear infinitely often in σ. We can easily see that there exist integers l, m and $\sigma_0(l) \in \mathcal{T}_G$ such that $f^i(0) \leq l < f^{j_m}(0)$ holds. By (3.4), $\sigma_0(l) \sqsupseteq_a \sigma_i(l)$, and thus $\sigma_i(l) \in \mathcal{T}_G$ because $\mathcal{T}_G$ is a downward-closed set. On the other hand, since $t_{\mathrm{R}} = \sigma_i(f^i(0)) \to^*_{\mathcal{R}} \sigma_i(l) \to^+_{\mathcal{R}} \sigma_i(f^{j_m}(0))$, we can obtain $\sigma_i(l) \in \mathcal{T}_G \cap pre^+(\{t_{\mathrm{R}}\vec{\theta}\})$ and $t_{\mathrm{R}} \in pre^*(\{\sigma_i(l)\})$ in a similar way to the above case. Hence, $t_{\mathrm{R}} \in pre^*(\mathcal{T}_G \cap pre^+(\{t_{\mathrm{R}}\vec{\theta}\}))$. □

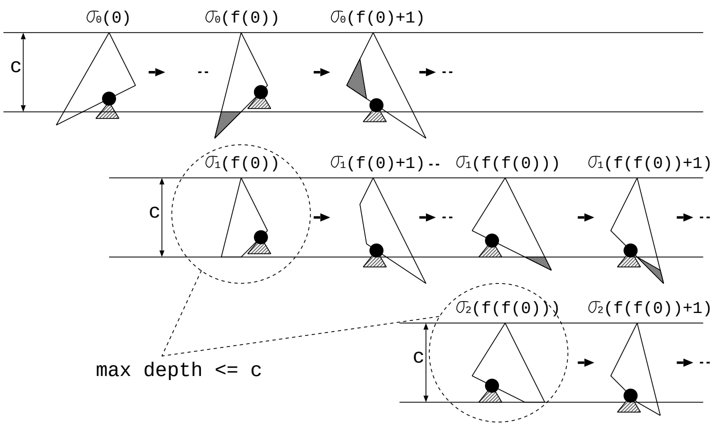

Figure 2: Proof of Lemma 3.3: infinite sequence $\sigma_0, \sigma_1, \cdots$

Theorem 3.4 *Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. Let $\mathcal{T}_G \subseteq \mathcal{T}(\mathcal{F} \cup \{a\})$ be a set of terms, which is upward-closed and downward-closed w.r.t $\sqsubseteq_a$. There exists an infinite root rewrite sequence of $\mathcal{R}$ starting in t_0 in* *which terms in $\mathcal{T}_G$ appear infinitely often if and only if there exist $t_{\mathrm{R}} \in \mathcal{T}(\mathcal{F} \cup \{a\})(|t_{\mathrm{R}}|_a = n)$ of which the depth is c or less and tuples of terms $\vec{\theta} \in \mathcal{T}^n(\mathcal{F} \cup \{a\}), \vec{\theta'} \in \mathcal{T}^n(\mathcal{F})$ such that $t_{\mathrm{R}} \in pre^*(\mathcal{T}_G \cap pre^+(\{t_{\mathrm{R}}\vec{\theta}\}))$ and $t_0 \in pre^*(\{t_{\mathrm{R}}\vec{\theta'}\})$ (or equivalently, $t_{\mathrm{R}}\vec{\theta} \in post^+(post^*(\{t_{\mathrm{R}}\}) \cap \mathcal{T}_G)$ and $t_{\mathrm{R}}\vec{\theta'} \in post^*(\{t_0\})$) hold.*

Proof. The *only if* part of this theorem follows from Lemma 3.3.

The *if* part is proved as follows. If $t_0 \in pre^*(\{t_{\mathrm{R}}\vec{\theta'}\})$ and $t_{\mathrm{R}} \in pre^*(\mathcal{T}_G \cap pre^+(\{t_{\mathrm{R}}\vec{\theta}\}))$, then there exists a term $t_{\mathrm{G}} \in \mathcal{T}_G$ such that $t_{\mathrm{R}} \in pre^*(\{t_{\mathrm{G}}\})$ and $t_{\mathrm{G}} \in pre^+(\{t_{\mathrm{R}}\vec{\theta}\})$ hold. By these facts, we can construct infinite root rewrite sequence $t_0 \to^*_{\mathcal{R}} t_{\mathrm{R}}\vec{\theta'} \to^*_{\mathcal{R}} t_{\mathrm{G}}\vec{\theta'} \to^+_{\mathcal{R}} t_{\mathrm{R}}\vec{\theta}\vec{\theta'} \to^*_{\mathcal{R}} t_{\mathrm{G}}\vec{\theta}\vec{\theta'} \to^+_{\mathcal{R}} t_{\mathrm{R}}\vec{\theta}^2\vec{\theta'} \to^*_{\mathcal{R}} \cdots$, where $\vec{\theta'} = \langle \theta'_1, \ldots, \theta'_n \rangle$ and $\vec{\theta}\vec{\theta'}$ is a term obtained by replacing a in $\vec{\theta}$ by one of $\theta'_1, \ldots, \theta'_n$. Since $t_{\mathrm{G}}\vec{\theta}^n\vec{\theta'} \sqsupseteq_a t_{\mathrm{G}}$ and $\mathcal{T}_G$ is upward-closed, $t_{\mathrm{G}}\vec{\theta}^n\vec{\theta'} \in \mathcal{T}_G$. Therefore, terms in $\mathcal{T}_G$ appear infinitely often in the above sequence. □

Theorem 3.5 *Let $\mathcal{R}$ be an LL-GG-TRS, $t_0 \in \mathcal{T}(\mathcal{F})$, ϕ be an LTL formula, ν be a simple valuation. There exists a term $t_{\mathrm{R}} \in \{q(t') \mid q \in \mathcal{Q}_{\mathcal{B}}, t' \in \mathcal{T}(\mathcal{F} \cup \{a\})\}$ of which the depth is c or less, and*

$$\begin{aligned} \mathcal{R}, t_0 \not\models^\nu \phi \quad &\Leftrightarrow \quad t_{\mathrm{R}} \in pre^*_{[\mathcal{BR}^\nu]}(\mathcal{T}_{\mathrm{acc}} \cap pre^+_{[\mathcal{BR}^\nu]}(\mathcal{T}'_{\mathrm{R}})) \text{ and } q_{0\mathcal{B}}(t_0) \in pre^*_{[\mathcal{BR}^\nu]}(\mathcal{T}_{\mathrm{R}}) \\ &\Leftrightarrow \quad post^+_{[\mathcal{BR}^\nu]}(post^*_{[\mathcal{BR}^\nu]}(\{t_{\mathrm{R}}\}) \cap \mathcal{T}_{\mathrm{acc}}) \cap \mathcal{T}'_{\mathrm{R}} \neq \emptyset \\ &\qquad \text{and } \mathcal{T}_{\mathrm{R}} \cap post^*_{[\mathcal{BR}^\nu]}(\{q_{0\mathcal{B}}(t_0)\}) \neq \emptyset, \end{aligned}$$

where $\mathcal{B}$ is a Büchi automaton representing $\neg\phi$, $q_{0\mathcal{B}}$ and $\mathcal{Q}^{acc}_{\mathcal{B}}$ are the initial state and accepting states of $\mathcal{B}$, $\mathcal{T}_{\mathrm{acc}} = \{q_{\mathrm{a}}(t) \mid q_{\mathrm{a}} \in \mathcal{Q}^{acc}_{\mathcal{B}}, t \in \mathcal{T}(\mathcal{F} \cup \{a\})\}$, $\mathcal{T}_{\mathrm{R}} = \{t_{\mathrm{R}}\vec{\theta} \mid \vec{\theta} \in \mathcal{T}^n(\mathcal{F})\}$, $\mathcal{T}'_{\mathrm{R}} = \{t_{\mathrm{R}}\vec{\theta} \mid \vec{\theta} \in \mathcal{T}^n(\mathcal{F} \cup \{a\})\}$.

Proof. If $\mathcal{R}$ is an LL-GG-TRS, then $\mathcal{BR}^\nu$ is also an LL-GG-TRS. $\mathcal{T}_{\mathrm{acc}}$ is upward-closed and downward-closed w.r.t $\sqsubseteq_a$. Therefore, by Lemma 2.2 and Theorem 3.4, the theorem holds. □

Corollary 3.6 *Let $\mathcal{R}$ be an LL-GG-TRS, $t_0 \in \mathcal{T}(\mathcal{F})$, ϕ be an LTL formula, ν be a simple valuation. If $\mathcal{BR}^\nu$ is pre-PR or post-PR, then $\mathcal{R}, t_0 \models^\nu \phi$ is decidable.*

Proof. The corollary follows from the facts that the number of candidates for t_{R} in Theorem 3.5 is finite, that we can construct TAs which recognize $\mathcal{T}_{\mathrm{acc}}$,

$\{q_{0\mathcal{B}}(t_0)\}$, $\mathcal{T}_{\mathrm{R}}$ and $\mathcal{T}'_{\mathrm{R}}$, and Lemma 2.1. $\square$

4 Computing pre^*

By Corollary 3.6, if an LL-GG-TRS $\mathcal{R}$ is post-PR or pre-PR, then LTL model checking for $\mathcal{R}$ is decidable. Unfortunately, an LL-GG-TRS is not always post-PR. For example, $\mathcal{R} = \{f(x,y) \to f(g(x), g(y))\}$ is an LL-GG-TRS. However, $post^+_{\mathcal{R}}(\{f(a,a)\})$ is not recognizable and thus $\mathcal{R}$ is not post-PR. It is unknown whether every LL-GG-TRS is pre-PR. In this section, we propose a decidable subclass of pre-PR TRS. Let $\mathcal{R}$ be a TRS. By the definition of pre-PR, for a given TA $\mathcal{A}$, if we can extend $\mathcal{A}$ so that $t \to_{\mathcal{R}} s \in pre^*_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$ implies $t \in pre^*_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$ (backward closedness w.r.t.$\to_{\mathcal{R}}$) then $\mathcal{R}$ is pre-PR. This requires us to add to $\mathcal{A}$ new states and transition rules to satisfy the condition that $t \to_{\mathcal{R}} s \vdash^*_{\mathcal{A}} q$ implies $t \vdash^*_{\mathcal{A}} q$. For example, let $f(g(x,y)) \to g(h(y),x) \in \mathcal{R}$, $t = f(g(a,b))$, $s = f(h(b),a)$, and $s \vdash^*_{\mathcal{A}} f(h(q_2),q_1) \vdash^*_{\mathcal{A}} q$ for states q_1, q_2 and q of a TA $\mathcal{A}$. Note that $t \to_{\mathcal{R}} s$ with substitution $\theta = \{x \mapsto a, y \mapsto b\}$. Then, we add the following states and transition rules to $\mathcal{A}$ so that $t \vdash^*_{\mathcal{A}} q$.

states: $\langle g(q_1,q_2)\rangle$, $\langle f(g(q_1,q_2))\rangle$.
rules: $g(q_1,q_2) \to \langle g(q_1,q_2)\rangle$, $f(\langle g(q_1,q_2)\rangle) \to \langle f(g(q_1,q_2))\rangle$, $\langle f(g(q_1,q_2))\rangle \to q$.

That is, we use a subterm of the left-hand side of the rewrite rule as a state to keep track of the position where the head of $\mathcal{A}$ is located. However, states substituted into variables such as q_1, q_2, q above may recursively be subterms, and hence the above construction does not always halt. The condtion for a TRS $\mathcal{R}$ to be an LL-SPO-TRS stated below is a sufficient condition for $\mathcal{R}$ not to have a kind of overlapping between subterms of rewrite rules, which guarantees that the above construction always halts.

4.1 LL-SPO-TRS

For an ordinary rewrite relation not limited to root rewriting, LL-FPO^{-1}-TRS is known as a decidable subclass of pre-PR TRS (see section 2.2). Based on the definition of LL-FPO^{-1}-TRS, we define a new subclass called LL-SPO-TRS and show that every LL-SPO-TRS is pre-PR with respect to root rewriting.

Definition 4.1 (**Sticking out relation**)
Let s and t be terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$. We say s sticks out of t if $t \notin \mathcal{V}$ and there exists

a position $o_{var} \in Pos(t)$ $(lab(t, o_{var}) \in \mathcal{V})$ such that

- for any positions o $(\lambda \preceq_{pref} o \prec_{pref} o_{var})$, $o \in Pos(s)$ and $lab(s, o) = lab(t, o)$, and
- $o_{var} \in Pos(s)$ and $s_{|o_{var}}$ is not a ground term.

When the position o_{var} is of interest, we say that s sticks out of t at o_{var}. If s sticks out of t at o_{var} and $lab(s, o_{var})$ is not a variable, then we say that s properly sticks out of t (at o_{var}). $\Box$

For example, $f(g(x), a)$ sticks out of $f(g(y), b)$ at 11 and $f(g(g(x)), a)$ properly sticks out of $f(g(y), b)$ at 11. Remember that a configuration of a PDS is a pair $\langle q, w \rangle$ of a control location (finite control) q and a sequence w of symbols stored in the pushdown stack. In the rest of this section, we assume that a signature $\mathcal{F}$ is decomposed into Π and Σ, that is, $\mathcal{F} = \Pi \cup \Sigma$ and $\Pi \cap \Sigma = \emptyset$. For each $\pi \in \Pi$, we assume $arity(\pi) = 1$. Each $\pi \in \Pi$ is called a control symbol and each $f \in \Sigma$ is called a data symbol.

Definition 4.2 (**Simply Path Overlapping TRS (SPO-TRS)**)
A TRS $\mathcal{R}$ is SPO if every rule in $\mathcal{R}$ has the form either $\pi_1(l) \to \pi_2(r)$, $\pi_1(l) \to r$ or $l \to r$ where $\pi_1, \pi_2 \in \Pi$ and $l, r \in \mathcal{T}(\Sigma, \mathcal{V})$, and the sticking-out graph $G_{\mathcal{R}}$ of $\mathcal{R}$ has no cycle with weight one or more. The sticking-out graph of a TRS $\mathcal{R}$ is a weighted directed graph $G_{\mathcal{R}} = (\mathcal{R}, E)$. Let $v_1 \xrightarrow{i} v_2$ denote a directed edge from a node v_1 to a node v_2 with weight i. E is defined as follows. Let $v_1 : l_1 \to r_1$ (or $\pi_{11}(l_1) \to r_1$ or $\pi_{11}(l_1) \to \pi_{12}(r_1)$) and $v_2 : l_2 \to r_2$ (or $\pi_{21}(l_2) \to r_2$ or $\pi_{21}(l_2) \to \pi_{22}(r_2)$) be rules in $\mathcal{R}$. Replace each variable in $\mathcal{V}ar(l_1) \backslash \mathcal{V}ar(r_1)$ or $\mathcal{V}ar(l_2) \backslash \mathcal{V}ar(r_2)$ with a constant not in $\mathcal{F}$, say $\diamond$.

(1) If l_1 properly sticks out of r_2, then $v_1 \xrightarrow{1} v_2 \in E$.

(2) If r_2 sticks out of l_1, then $v_1 \xrightarrow{0} v_2 \in E$.

$\Box$

If $\mathcal{R}$ is an LL-SPO-TRS, then for any TA $\mathcal{A}$, we can construct a TA $\mathcal{A}_*$ such that $\mathcal{L}(\mathcal{A}_*) = pre^*_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$ (see Appendix). That is, every LL-SPO-TRS is pre-PR.

Theorem 4.3 *For every recognizable tree language L and LL-SPO-TRS $\mathcal{R}$, $pre^*_{\mathcal{R}}(L)$ is also recognizable.* □

Corollary 4.4 *Assume $t_0 \in \mathcal{T}(\mathcal{F})$, ϕ is an LTL formula and ν is a simple valuation. If $\mathcal{R} \in$ LL-GG-TRS $\cap$ SPO-TRS, then $\mathcal{R}, t_0 \models^\nu \phi$ is decidable.*

Proof. Let $\mathcal{B} = (\mathcal{Q}_\mathcal{B}, \Sigma_\mathcal{B}, \Delta_\mathcal{B}, q_{0\mathcal{B}}, \mathcal{Q}^{acc}_\mathcal{B})$ be a Büchi automaton representing $\neg\phi$ and Π be the set of control symbols of $\mathcal{R}$. Consider the construction of Büchi TRS $\mathcal{BR}^\nu$ from $\mathcal{R}$, $\mathcal{B}$ and ν. If $\mathcal{R}$ is an SPO-TRS, then by constructing $\langle q, p\rangle(l) \to \langle q', p'\rangle(r) \in \mathcal{BR}^\nu$ instead of $q(p(l)) \to q'(p'(r)) \in \mathcal{BR}^\nu$ for each rule $p(l) \to p'(r) \in \mathcal{R}$ $(p, p' \in P)$, $\mathcal{BR}^\nu$ becomes an SPO-TRS. By Corollary 3.6 and Theorem 4.3, $\mathcal{R}, t_0 \models^\nu \phi$ is decidable. □

4.2 Application

As mentioned in section 3, we can model a recursive program with exception handling by an LL-GG-TRS. If the LL-GG-TRS is always an SPO-TRS, then LTL model checking for the TRS is decidable. Recall $\mathcal{R}_2$ in Example 3.2. Since for any two rules $l_1 \to r_1$ and $l_2 \to r_2$ in $\mathcal{R}_2$, l_1 never properly sticks out of r_2, $\mathcal{R}_2$ is an SPO-TRS. Similarly, we can easily see that every LL-GG-TRS constructed by the method in Example 3.2 is always an SPO-TRS. Thus, LTL model checking problem is decidable for recursive programs with exception handling.

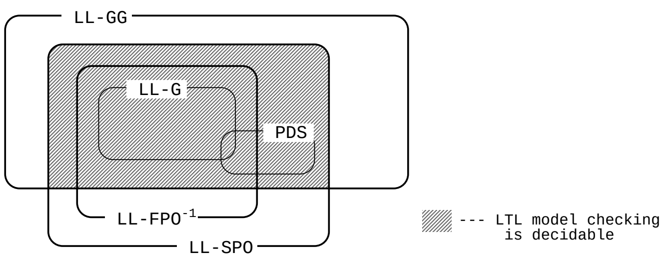

Figure 3: The relation between TRS subclasses

5 Conclusion

In this paper, we introduced two classes of TRS, LL-GG-TRS and SPO-TRS, and showed that for a TRS in LL-GG-TRS $\cap$ SPO-TRS, LTL model checking is decidable. Since every PDS is a member of LL-GG-TRS $\cap$ SPO-TRS, this model checking is considered as an extension of LTL model checking for PDS. In fact, a recursive program with exception handling can be modeled as a TRS to which this model checking method can be applied and to which no PDS is strongly bisimilar.

We can reduce some decision problems of TRS to LTL model checking problems. For example, let ν be a regular valuation and α_{NF} be an atomic proposition such that $\nu(\alpha_{NF}) = \mathrm{NF}_{\mathcal{R}}$. Whether there exists no infinite rewrite sequence starting in t_0 (strongly normalizing) is checked by $\mathcal{R}, t_0 \models^{\nu} \Diamond(\alpha_{NF})$, and whether there exists a finite rewrite sequence starting in t_0 (weakly normalizing) is checked by $\mathcal{R}, t_0 \not\models^{\nu} \Box(\neg\alpha_{NF})$.

The following problems remain as future study:

- finding a wider subclass of TRS in which LTL model checking is solvable,
- developing an efficient LTL model checking method w.r.t. regular valuation,
- and finding other applications of this model checking method.

Acknowledgments The authors would like to thank Professor Igor Walukiewicz of Université Bordeaux I for his valuable comments and discussions.

References

[1] F. Baader and T. Nipkow: *Term Rewriting and All That*, Cambridge University Press, 1998.

[2] E. M. Clarke, O. Grumberg and D. Peled: *Model Checking,* MIT Press, 2000.

[3] J. L. Coquidé, M. Dauchet, R. Gilleron and S. Vágvölgyi: Bottom-up tree pushdown automata: classification and connection with rewrite systems, *Theoretical Computer Science*, **127**, 69–98, 1994.

[4] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, Robby and H. Zheng: Bandera: Extracting finite-state models from Java source code, Int'l Conf. on Software Engineering, 439–448, 2000.

[5] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon: Efficient algorithms for model-checking pushdown systems, CAV2000, LNCS **1855**, 232–247, 2000.

[6] J. Esparza, A. Kučera and S. Schwoon: Model-checking LTL with regular variations for pushdown systems, TACS01, LNCS **2215**, 316–339, 2001.

[7] J. Esparza and S. Schwoon: A BDD-based model checker for recursive programs, CAV2001, LNCS **2102**, 324–336, 2001.

[8] F. Gécseq and M. Steinby: *Tree Automata*, Académiai Kiadó, 1984.

[9] R. Gilleron: Decision problems for term rewriting systems and recognizable tree languages, STACS'91, LNCS **480**, 148–159, 1991.

[10] R. Gilleron and S. Tison: Regular tree languages and rewrite systems, *Fundamenta Informaticae*, **24**, 157–175, 1995.

[11] L. Gong, M. Mueller, H. Prafullchandra and R. Schemers: Going beyond the sandbox: An overview of the new security architecture in the Java^{TM} development kit 1.2, USENIX Symp. on Internet Technologies and Systems, 103–112, 1997.

[12] S. Graf and H. Saïdi: Construction of abstract state graphs with PVS, CAV97, LNCS **1254**, 72–83, 1997.

[13] P. Gyenizse and S. Vágvölgyi: Linear generalized semi-monadic rewrite systems effectively preserve recognizability, *Theoretical Computer Science*, **194**, 87–122, 1998.

[14] T. Jensen, D. Le Métayer and T. Thorn: Verification of control flow based security properties, IEEE Symp. on Security and Privacy, 89–103, 1999.

[15] S. Jha and T. Reps: Analysis of SPKI/SDSI certificates using model checking, IEEE Computer Security Foundations Workshop, 129–144, 2002.

[16] C. Löding: Model-checking infinite systems generated by ground tree rewriting, FOSSACS, LNCS **2303**, 280–294, 2002.

[17] R. Mayr: Process rewrite systems, *Inform. & Comput.*, **156**, 264–286, 1999.

[18] T. Nagaya and Y. Toyama: Decidability for left-linear growing term rewriting systems, RTA99, LNCS **1631**, 256–270, 1999.

[19] N. Nitta, Y. Takata and H. Seki: Security verification of programs with stack inspection, 6th ACM Symp. on Access Control Models and Technologies, 31–40, 2001.

[20] N. Nitta, Y. Takata and H. Seki: An efficient security verification method for programs with stack inspection, 8th ACM Conf. on Computer and Communication Security, 68–77, 2001.

[21] J. Obdržálek: Model checking Java using pushdown systems, ECOOP Workshop on Formal Techniques for Java-like Programs, 2002.

[22] `http://www.pobox.com/~cme/spki.txt`

[23] T. Takai, Y. Kaji and H. Seki: Right-linear finite path overlapping term rewriting systems effectively preserve recognizability, RTA2000, LNCS **1833**, 246–260, 2000.

[24] I. Walukiewicz: Pushdown processes: Games and model-checking, CAV96, LNCS **1102**, 62–74, 1996.

Appendix

A.1 Proof of Lemma 3.1(a)

Lemma 3.1 *(a) Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. Also let $\sigma = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_{k-1} \to_{\mathcal{R}} t_k \to_{\mathcal{R}} t_{k+1}$ of $\mathcal{R}$ be a root rewrite sequence and $o_0 \in Pos(t_0)$ be a position. If there exists $o_i \in Res(o_0, t_i)$ such that $|o_0| \leq |o_i|$ for each $i (1 \leq i \leq k)$, then every position $p_0 \in Pos_{\geq c}(t_0)$ satisfies:*

For an arbitrary $p_k \in Res(p_0, t_k)$,

$$|p_k| - |o_k \sqcup p_k| \geq |p_0| - |o_0 \sqcup p_0| \ \text{ and } \ |p_k| > max_f(\mathcal{R}).$$

Proof. The proof is shown by induction on the length k of σ (Fig.4).

- $k = 0$: $|p_k| = |p_0| \geq c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1 > max_f(\mathcal{R})$.
- $k \geq 1$: Let $p_j \in Res(p_{j-1}, t_j)$ $(1 \leq j \leq k)$, o_j be the deepest residual of o_{j-1} in term t_j. By the induction hypothesis, $|p_{k-1}| - |o_{k-1} \sqcup p_{k-1}| \geq |p_0| - |o_0 \sqcup p_0|$. Since $\mathcal{R}$ is generalized-growing, $|p_k| - |o_k \sqcup p_k| \geq |p_{k-1}| - |o_{k-1} \sqcup p_{k-1}|$. Consequently,
$$|p_k| - |o_k \sqcup p_k| \geq |p_0| - |o_0 \sqcup p_0|. \tag{A.1}$$
Next, we show $|p_k| > max_f(\mathcal{R})$ according to the following two cases.
  - $|o_k \sqcup p_k| \geq |o_0 \sqcup p_0|$:
By (A.1),
$$|p_k| \geq |p_0| \geq c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1 > max_f(\mathcal{R}). \tag{A.2}$$
  - $|o_k \sqcup p_k| < |o_0 \sqcup p_0|$:
Since $|o_k| \geq |o_0|$ by the hypothesis of the lemma, $|o_k| - |o_k \sqcup p_k| > |o_0| - |o_0 \sqcup p_0|$. Therefore, there exists $i (1 \leq i \leq k)$ such that
$$|o_i| - |o_i \sqcup p_i| > |o_{i-1}| - |o_{i-1} \sqcup p_{i-1}| = |o_0| - |o_0 \sqcup p_0|. \tag{A.3}$$
Let $l_{i-1} \to r_{i-1}$ be the rule applied in $t_{i-1} \to_{\mathcal{R}} t_i$, then by (A.3), there exists a variable v in l_{i-1} and its occurrence o_v satisfies $o_{i-1} \sqcup p_{i-1} \prec_{pref} o_v \preceq_{pref} o_{i-1}$. Hence,
$$|o_{i-1} \sqcup p_{i-1}| < |o_v| \leq max_v(\mathcal{R}). \tag{A.4}$$

Again, by the hypothesis of the lemma, $|o_0| \leq |o_{i-1}|$. Therefore, by (A.3),

$$|o_0 \sqcup p_0| = |o_0| - |o_{i-1}| + |o_{i-1} \sqcup p_{i-1}| \leq |o_{i-1} \sqcup p_{i-1}|. \tag{A.5}$$

By (A.4) and (A.5), we have

$$|o_0 \sqcup p_0| < max_v(\mathcal{R}). \tag{A.6}$$

Also, by the hypothesis of the lemma,

$$|p_0| \geq c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1. \tag{A.7}$$

Finally, by (A.1),(A.6) and (A.7),

$$\begin{aligned} |p_k| &\geq |o_k \sqcup p_k| + |p_0| - |o_0 \sqcup p_0| \\ &\geq max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1 - |o_0 \sqcup p_0| \\ &> max_f(\mathcal{R}) \end{aligned}$$

holds.

□

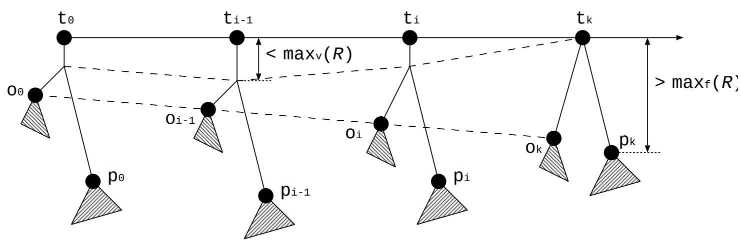

Figure 4: Proof of Lemma 3.1: The depth of p_k

A.2 Proof of Lemma 3.2

Lemma 3.2 *Let $\mathcal{R}$ be an LL-GG-TRS and $c = max_v(\mathcal{R}) + max_f(\mathcal{R}) + 1$. If there exists an infinite root rewrite sequence $\sigma = t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} \cdots$ of $\mathcal{R}$, then for* *any $n \geq 0$, there exists $m > n$ such that for every $p_m \in Pos_{\geq c}(t_m)$, $k > m$ and $s \in \mathcal{T}(\mathcal{F})$:*

$$t_m[p_m \leftarrow s] \rightarrow^*_{\mathcal{R}} t_k[Res(p_m, t_k) \leftarrow s].$$

Proof. There are two cases.

- Assume that there exists a position p_n in t_n of which the lifetime is undefined. Let p_i be the deepest residual of p_n in $t_i (i > n)$, and m be the minimum $j (> n)$ such that $|p_j| \leq |p_i|$ for each $i (> j)$. Note that m is always defined since $t_n \rightarrow_{\mathcal{R}} t_{n+1} \rightarrow_{\mathcal{R}} \ldots$ is an infinite sequence. Also, $t_m \rightarrow_{\mathcal{R}} t_{m+1} \rightarrow_{\mathcal{R}} \ldots$ and p_m satisfy the hypothesis of Lemma 3.1. Hence, by Lemma 3.1(b), the lemma holds.

- Assume that the lifetime of every position in t_n is defined and let p^0_n be the longest-living position of t_n, and τ be the lifetime of p^0_n. For each term $t_i (n < i \leq n + \tau)$, let p^0_i be the deepest residual of p^0_n in t_i, and m_0 be the minimum $j (> n)$ such that $|p^0_j| \leq |p^0_i|$ for each $i (j < i \leq n + \tau)$. Note that if $\tau = 0$, then $m_0 = n+1$ and if $\forall_{i(n+1<i\leq n+\tau)}.|p^0_{i-1}| > |p^0_i|$, then $m_0 = n+\tau$. An integer m and a position o are calculated by the following recursive procedure. The procedure starts with $i := 0$ and at each recursion step, let p^{i+1} be a position in $Pos_{\geq c}(t_{m_i})$, which has the maximum lifetime in $Pos_{\geq c}(t_{m_i})$.

  - The lifetime of p^{i+1} is $n + \tau - m_i$ or less:
    $m := m_i$, $o := p^i_{m_i}$ and the procedure terminates.
  - The lifetime of p^{i+1} is more than $n + \tau - m_i$:
    In this case, there is no position in t_n which has p^{i+1} as a residual since the lifetime of the longest-living position in t_n is τ. Let $t_{n'}$ be the first term in σ containing a position p' which has p^{i+1} as a residual (i.e., $\exists_{p' \in Pos(t_{n'})}.p^{i+1} \in Res(p', t_{n'} \rightarrow^*_{\mathcal{R}} t_{m_i})$). Let p^{i+1}_j be a position in t_j which has p^{i+1} as a residual for $n' \leq j \leq m_i$, and be the most shallow residual of p^{i+1} in t_j for $m_i < j \leq n + \tau$. Note that $p^{i+1}_{m_i} = p^{i+1}$. Since $p^{i+1}_{n'}$ is created by $t_{n'-1} \rightarrow_{\mathcal{R}} t_{n'}$,

    $$|p^{i+1}_{n'}| \leq max_f(\mathcal{R}). \tag{A.8}$$

In this procedure, we assume the induction hypothesis on i: For $j(m_i < j \leq n+\tau)$, $|p^i_{m_i}| \leq |p^i_j|$. Remember that $|p^{i+1}_{m_i}| = |p^{i+1}| \geq c$. Hence by Lemma 3.1(a),

$$|p^{i+1}_j| > max_f(\mathcal{R}). \tag{A.9}$$

Let m_{i+1} be the minimum $j(n' \leq j \leq m_i)$ such that $\forall_{j'(n' \leq j < j' \leq m_i)}.|p^{i+1}_j| \leq |p^{i+1}_{j'}|$. Then, by (A.8) and (A.9), for any $j(n' \leq j \leq n+\tau)$, $|p^{i+1}_{m_{i+1}}| \leq |p^{i+1}_j|$ holds (Fig.5). Finally, let $i := i+1$.

Consider the case of $m_i = m_{i+1} = m_{i+2} = \ldots$. Since the life-time of $p^i_{m_i}$ increases with i, $p^i_{m_i}, p^{i+1}_{m_{i+1}}, p^{i+2}_{m_{i+2}}, \ldots$ are different positions in t_{m_i}. Therefore, from the fact that $n < m_{i+1} \leq m_i$, the procedure always terminates. In any case, there is no position $p' \in Pos_{\geq c}(t_m)$ of which the lifetime is more than $n + \tau - m$. Thus, for an arbitrary position $p' \in Pos_{\geq c}(t_m)$, $Res(p', t_k) = \emptyset$ $(k \geq n+\tau+1)$ holds. On the other hand, for each $j(m < j \leq n+\tau)$, there exists $o'_j \in Res(o, t_j)$ and $|o| \leq |o'_j|$ holds by the above procedure. Therefore, by Lemma 3.1(b), $t_m[p' \leftarrow s] \rightarrow^*_{\mathcal{R}} t_j[Res(p', t_j) \leftarrow s] \rightarrow^*_{\mathcal{R}} t_{n+\tau+1}[Res(p', t_{n+\tau+1}) \leftarrow s] = t_{n+\tau+1} \rightarrow^*_{\mathcal{R}} t_k = t_k[Res(p', t_k) \leftarrow s]$ $(m < j \leq n+\tau, k > n+\tau+1)$ and the lemma holds.

□

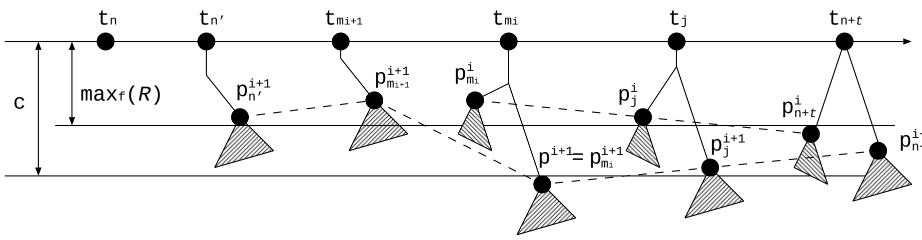

Figure 5: Proof of Lemma 3.2: Selection of p^{i+1} and m_{i+1}

A.3 Constructing $\mathcal{A}_*$

Construction 1

For simplicity, we first consider the case that $P = \emptyset$.

Input: Left-linear TRS $\mathcal{R}$
Deterministic and complete TA $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \Delta, \mathcal{Q}^{final})$

Output: $\mathcal{A}_*$ such that $\mathcal{L}(\mathcal{A}_*) = pre^*_{\mathcal{R}}(\mathcal{L}(\mathcal{A}))$

Procedure: In the following steps, a series of TA $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots$ are constructed. Let $\mathcal{A}_k = (\mathcal{F}_k, \mathcal{Q}_k, \Delta_k, \mathcal{Q}_k^{final})$ and $\vdash_{\mathcal{A}_k}$ is abbreviated as $\vdash_k$.

Step1: $\mathcal{A}_0 := (\mathcal{F}, \mathcal{Q} \cup \tilde{\mathcal{Q}}^{final}, \Delta, \mathcal{Q}^{final} \cup \tilde{\mathcal{Q}}^{final})$ where $\tilde{\mathcal{Q}}^{final} = \{\tilde{q} \mid q \in \mathcal{Q}^{final}\}$; $k := 0$;

Step2: $\mathcal{Q}_{k+1} := \mathcal{Q}_k$, $\mathcal{Q}_{k+1}^{final} := \mathcal{Q}_k^{final}$, $\Delta_{k+1} := \Delta_k$;

Step3:

$$\frac{\begin{array}{c} l \to r \in \mathcal{R}, \\ \mathcal{V}ar(r) = \{x_1, \ldots, x_m\}, \\ x_i (1 \le i \le m) \text{ occurs at } o_{i1}, \ldots, o_{i\gamma_i} \text{ in } r, \\ r[o_{ij} \leftarrow q_{ij} \in \mathcal{Q}_k | 1 \le i \le m, 1 \le j \le \gamma_i] \vdash_k^* q \in \mathcal{Q}_k^{final} \\ \text{where no } \epsilon\text{-move occurs at a variable position of } r, \\ \bot \ne t_i = \bigsqcup\{q_{ij} | 1 \le j \le \gamma_j\} \ (1 \le i \le m), \\ \rho' : \mathcal{V}ar(l) \setminus \mathcal{V}ar(r) \to \mathcal{Q}_k, \\ \rho = \{x_i \mapsto t_i | 1 \le i \le m\} \cup \rho' \end{array}}{\begin{array}{c} \langle l\rho \rangle \to \tilde{q} \in \Delta_{k+1}, \\ ADDREC(l, \rho), \\ \langle t_i \rangle \in \mathcal{Q}_{k+1} (1 \le i \le m). \end{array}} \tag{A.10}$$

$$\frac{ADDREC(f(t_1, \ldots, t_n), \rho)}{\begin{array}{c} f(\langle t_1\rho \rangle, \ldots, \langle t_n\rho \rangle) \to \langle f(t_1, \ldots, t_n) \rangle \in \Delta_{k+1}, \\ ADDREC(t_i, \rho)(1 \le i \le n), \\ \langle f(t_1, \ldots, t_n) \rangle \in \mathcal{Q}_{k+1}. \end{array}} \tag{A.11}$$

In (A.10) and (A.11), if $q = \tilde{q'}$ for $q' \in \mathcal{Q}_k^{final}$, then $\tilde{q}$ denotes q itself. Also, if $t \in \mathcal{Q}_k$, then $\langle t \rangle$ denotes t. For example, if $q_0 \in \mathcal{Q}_0$, then $\langle q_0 \rangle = q_0$, and if $t = \langle f(q_1) \rangle \in \mathcal{Q}_k$, then $\langle t \rangle = t$.

Step4: If $\Delta_{k+1} = \Delta_k$, then output $\mathcal{A}_k$ as $\mathcal{A}_*$ and halt. Otherwise, $k := k + 1$ and goto Step2.

The operation $\sqcup$ used in (A.10) is defined as:

$$t \sqcup t' \equiv \begin{cases} t & \text{if } t = t' \in \mathcal{Q} \text{ or} \\ & t \in \mathcal{T}(\Sigma \cup \mathcal{Q}) \setminus \mathcal{Q}, t' \in \mathcal{Q}, t \vdash_0^* t' \\ t' & \text{if } t \in \mathcal{Q}, t' \in \mathcal{T}(\Sigma \cup \mathcal{Q}) \setminus \mathcal{Q}, t' \vdash_0^* t, \\ f(t_1 \sqcup t'_1, \ldots, t_n \sqcup t'_n) & \\ & \text{if } t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma \cup \mathcal{Q}) \setminus \mathcal{Q}, \\ & t' = f(t'_1, \ldots, t'_n) \in \mathcal{T}(\Sigma \cup \mathcal{Q}) \setminus \mathcal{Q}, \text{ and} \\ & t_i \sqcup t'_i \neq \bot (1 \leq i \leq n), \\ \bot & \text{otherwise.} \end{cases}$$

In the general case where $\Pi \neq \emptyset$, add to Step 3 a construction rule obtained from (A.10) by replacing $l \to r \in \mathcal{R}$ with $\pi(l) \to r \in \mathcal{R}$ ($\pi \in \Pi$, $l \in \mathcal{T}(\Sigma, \mathcal{V})$ and $r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$) in the premise and replacing $\langle l\rho \rangle \to \tilde{q} \in \Delta_{k+1}$ with $\langle l\rho \rangle \to [\pi, \tilde{q}]$, $\pi([\pi, \tilde{q}]) \to \tilde{q} \in \Delta_{k+1}$, $[\pi, \tilde{q}] \in \mathcal{Q}_{k+1}$ in the conclusion.

□

The above construction satisfies the following properties. The proofs are similar to those in [23] and omitted here.

Lemma 4.1 *(Soundness and Completeness)* $\exists_{k \geq 0} : t \vdash_k^* q$ *or* $\tilde{q}$ *(*$q \in \mathcal{Q}^{final}$*) if and only if* $t \to_{\mathcal{R}}^* {}^{\exists}s \vdash_0^* q$. □

Lemma 4.2 *(Termination) For every TA* $\mathcal{A}$ *and LL-SPO-TRS* $\mathcal{R}$*, Construction 1 halts.* □

By Lemmas 4.1 and 4.2, we can show that every LL-SPO-TRS is pre-PR.

A.4 Non-Bisimilarity between PDS and Program with Exception Handling

Consider the following LL-GG-TRS $\mathcal{R}_3$ with $\mathcal{F} = \{L, \Box\}$ ($arity(L) = 2, arity(\Box) = 0$):

$$
\begin{aligned}
\mathcal{R}_3 \ = \{ & \\
& L(x,y) \to x, & &: \textbf{ret} \\
& L(x,y) \to y, & &: \textbf{throw} \\
& L(x,y) \to L(L(x,y),y), & &: \textbf{call} \\
& L(x,y) \to L(x,L(x,y)) & &: \textbf{try-catch} \\
\}. &
\end{aligned}
$$

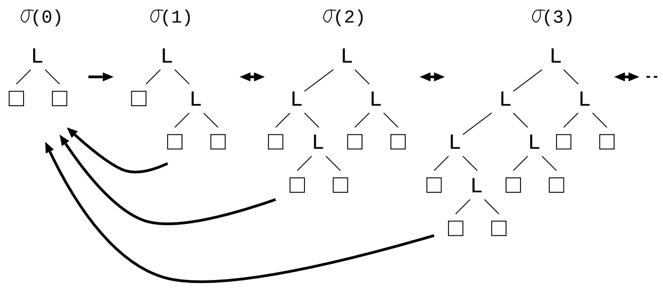

Figure 6: An infinite root rewrite sequence of $\mathcal{R}_3$

$\mathcal{R}_3$ models a program with exception handling and is an example of LL-GG-TRS having no strongly bisimilar PDS, which will briefly show here. (Let $L(\Box, \Box)$ be the initial state of $\mathcal{R}_3$ as a transition system.) Assume that there exists a PDS $\mathcal{P} = (P, \Gamma, \Delta, q_0)$ which is strongly bisimilar to $\mathcal{R}_3$, where P is a finite set of control locations, Γ is a finite stack alphabet, $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of transition rules, $q_0 \in P$ is an initial location. Note that the initial configuration of $\mathcal{P}$ is $\langle q_0, \lambda \rangle$. Let $R \subseteq \mathcal{T}(\mathcal{F}) \times (P \times \Gamma^*)$ be a strong bisimulation relation and $h : \mathcal{T}(\mathcal{F}) \to (P \times \Gamma^*)$ be a function such that $t \, R \, h(t)$ for any term t. Consider an infinite root rewrite sequence σ (Fig.6) such that $\sigma(0) = L(\Box, \Box)$, $\sigma(1) = L(\Box, L(\Box, \Box))$ (obtained by applying the **try-catch** rule), and $\sigma(n+1)$ is obtained by applying the **call** rule to $\sigma(n)$ for $n \geq 1$, where $\sigma(n)$ ($n \geq 0$) denotes the n-th term in σ. We can see that for any two different integers $n' > n > 0$, there exists no configuration $\langle q, w\rangle$ in $\mathcal{P}$ such that $\sigma(n) \, R \, \langle q, w\rangle$ and $\sigma(n') \, R \, \langle q, w\rangle$ by the following reason. Assume that there exists such a configuration $\langle q, w\rangle$. From bisimilarity of R, $\sigma(0)$ can be reached by n' times transitions from $\sigma(n)$. This is a contradiction since $\sigma(0)$ can be reached only by n or less transitions from $\sigma(n)$. Hence, $h(\sigma(n)) \neq h(\sigma(n'))$ whenever $n \neq n'$. Since σ is an infinite sequence, $\{h(\sigma(n)) \mid n \geq 0\}$ is also infinite. Let $\langle q_n, w_n\rangle = h(\sigma(n))$. Then, from the finiteness of P, we can see that $W = \{w_n \mid n \geq 0\}$ is infinite. On the other hand, $\sigma(n) \to_{\mathcal{R}_3} \sigma(0)$ for any $n > 0$. This implies $\langle q_n, w_n\rangle \to \langle q_0, \lambda\rangle \in \Delta$ and thus w_n is bounded. This contradicts the infinity of W.