On Reachability Analysis of Pushdown Systems with Transductions: Application to Boolean Programs with Call-by-Reference*

Fu Song, Weikai Miao, Geguang Pu, and Min Zhang†

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
{fsong,wkmiao,ggpu,mzhang}@sei.ecnu.edu.cn

Abstract

Pushdown systems with transductions (TrPDSs) are an extension of pushdown systems (PDSs) by associating each transition rule with a transduction, which allows to inspect and modify the stack content at each step of a transition rule. It was shown by Uezato and Minamide that TrPDSs can model PDSs with checkpoint and discrete-timed PDSs. Moreover, TrPDSs can be simulated by PDSs and the predecessor configurations $pre^*(C)$ of a regular set C of configurations can be computed by a saturation procedure when the closure of the transductions in TrPDSs is finite. In this work, we comprehensively investigate the reachability problem of finite TrPDSs. We propose a novel saturation procedure to compute $pre^*(C)$ for finite TrPDSs. Also, we introduce a saturation procedure to compute the successor configurations $post^*(C)$ of a regular set C of configurations for finite TrPDSs. From these two saturation procedures, we present two efficient implementation algorithms to compute $pre^*(C)$ and $post^*(C)$. Finally, we show how the presence of transductions enables the modeling of Boolean programs with call-by-reference parameter passing. The TrPDS model has finite closure of transductions which results in model-checking approach for Boolean programs with call-by-reference parameter passing against safety properties.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Verification, Reachability problem, Pushdown system with transductions, Boolean programs with call-by-reference

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.383

1 Introduction

A pushdown system (PDS) consists of a finite set of states and a finite stack alphabet, where stack can store context. PDS is one of the most widely used models for sequential programs with recursion [13, 25, 28]. Thanks to the efficient algorithms for the reachability problem of PDSs [6, 13], several software model-checking tools such as Moped [16], PDSolver [17], PuMoC [26] are implemented based the theory of PDSs for C/C++, Java and Boolean program verification. Several extensions of PDSs are proposed to model more complex behaviors.

* This work was partially supported by Shanghai Pujiang Program (No. 14PJ1403200), NSFC Projects (Nos. 61402179, 91418203,61202105,61361136002,91118007), Shanghai ChenGuang Program (No. 13CG21) and China HGJ Project (No. 2014ZX01038-101-001).

† Corresponding author

Various classes of pushdown automata such as multi-stack PDSs [9], pushdown networks [7, 18, 27, 29] and well-structure PDSs [10] have been proposed for modeling concurrent (thread-creation) programs with recursion. In order to model timed (resp. probabilistic) behavior, models that combine timed (resp. probabilistic) automata and PDSs are investigated in the literature, e.g., discrete/dense-timed pushdown systems [1, 2], nested timed automata [20] (resp. probabilistic PDSs [14, 8]). For dataflow analysis purpose, weighted PDSs [23] and extended weighted PDSs [19] are proposed, where transitions are associated with values from semirings. For stack manipulation of PDSs, Esparza et al. introduced PDSs with checkpoint [15] that can check the full stack content against a regular language (recognized by a finite state automaton) over the stack alphabet. This model is called conditional PDSs in [21] and transformable PDSs in [30]. Uezato and Minamide extended PDSs with transductions (TrPDSs) which associate each transition with a transduction. The associated transductions can check the stack content and modify the whole stack content. TrPDSs are a generalization of PDSs with checkpoint and discrete-timed PDSs. In general, TrPDSs are Turing complete. To achieve decidability result, Uezato and Minamide considered *finite* TrPDSs which restrict the closure of transductions appearing in the transitions of a TrPDS to be finite. They showed that a finite TrPDS can be simulated by a PDS. Therefore, the reachability problem of finite TrPDSs is decidable. Moreover, the saturation procedure that calculates the set $pre^*(C)$ of predecessor configurations for a given regular set of configurations C can be directly extended from PDSs to finite TrPDSs.

In this work, we follow the direction of [30] and make a comprehensive study of the reachability problem of TrPDSs. The main contributions of this paper can be summarized as follows:

- A novel saturation procedure is proposed that computes the set $pre^*(C)$ of predecessor configurations for a given regular set of configurations C of TrPDSs (cf. Section 3.1). This saturation procedure avoids *pseudo formal power series semiring* that was introduced in [30] to compute $pre^*(C)$.
- A saturation procedure is introduced to compute the set $post^*(C)$ of successor configurations for a given regular set of configurations C of TrPDSs (cf. Section 4.1). TrPDSs can be simulated by PDSs as shown in [30]. Therefore, $post^*(C)$ could be computed by applying the saturation procedure of PDSs [13]. Our saturation procedure directly computes a kind of finite state automaton that exactly recognizes $post^*(C)$. We believe our direct approach is more convenient for studying optimal algorithm or BDD-based symbolic techniques.
- Efficient implementation algorithms of the saturation procedures for computing $pre^*(C)$ and $post^*(C)$ are presented (cf. Section 3.2 and Section 4.2). We show that the computations of both $pre^*(C)$ and $post^*(C)$ are fixed-parameter tractable with the fixed-parameter of transductions.
- We show that TrPDSs are powerful enough to model Boolean programs with call-by-reference parameter passing. Boolean programs in the literature [3] only consider call-by-value parameter passing which can be modeled by PDSs. Using our approach, safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing can be directly verified (cf. Section 5).

Section 2 presents basic definitions. Section 3 (resp. Section 4) introduce the saturation procedure and its efficient implementation algorithm for computing $pre^*(C)$ (resp. $post^*(C)$). In Section 5, we present a potential application of TrPDSs for modeling and verifying Boolean programs with call-by-reference parameter passing. Section 6 discusses related work. Section 7 concludes and discusses future work. Due to space limitation, proofs and details of Examples (9 and 13) are omitted and will appear in the journal version of this paper.

2 Preliminaries

2.1 Finite-State Transducers and Transduction

▶ **Definition 1.** A *finite-state transducer* (FST) $\mathbf{T}$ is a tuple $(Q, \Gamma, \delta, I, F)$, where Q is a finite set of states, Γ is a finite alphabet, $\delta \subseteq Q \times \Gamma^* \times \Gamma^* \times Q$ is a finite set of transition rules, $I \subseteq Q$ (resp. $F \subseteq Q$) is a finite set of initial (resp. final) states. The transducer is *letter-to-letter* if $\delta \subseteq Q \times \Gamma \times \Gamma \times Q$.

We will write $q \xrightarrow{\omega_1/\omega_2} q'$ if $(q, \omega_1, \omega_2, q') \in \delta$. Let $\longrightarrow^*$ be the smallest relation such that $q \xrightarrow{\epsilon/\epsilon}{}^* q$ for every $q \in Q$; if $q \xrightarrow{\omega_1/\omega_2}{}^* q'$ and $q' \xrightarrow{\omega_3/\omega_4} q''$, then $q \xrightarrow{\omega_1\omega_3/\omega_2\omega_4}{}^* q''$. A FST $\mathbf{T}$ transduces a string $\omega_1 \in \Gamma^*$ into a string $\omega_2 \in \Gamma^*$ if there exist states $q_0 \in I$ and $q_f \in F$ such that $q_0 \xrightarrow{\omega_1/\omega_2}{}^* q_f$. The language $L(\mathbf{T})$ of a FST $\mathbf{T}$ is the set of pairs (ω_1, ω_2) such that $\mathbf{T}$ can transduce ω_1 into ω_2.

A *transduction* $\tau \subseteq \Gamma^* \times \Gamma^*$ is a relation over Γ^*. A transduction τ is *rational* (regular) and *length-preserving* if there is a letter-to-letter transducer $\mathbf{T}$ such that $\tau = L(\mathbf{T})$. Let τ_{id} denote the *identity transduction*, i.e., $\tau_{id} = \{(\omega, \omega) \mid \forall \omega \in \Gamma^*\}$. In the rest of this paper, we assume that transductions (resp. transducers) are length-preserving rational (resp. letter-to-letter) unless stated explicitly, and we do not differentiate the terms transduction and transducer. Given a transduction τ, let $\tau(\omega) = \{\omega' \mid (\omega, \omega') \in \tau\}$ for $\omega \in \Gamma^*$.

The *composition* $\circ$ of two transductions τ_1, τ_2 is defined as

$$\tau_1 \circ \tau_2 = \{(\omega_1, \omega_3) \mid \exists \omega_2 \in \Gamma^*, (\omega_1, \omega_2) \in \tau_1 \text{ and } (\omega_2, \omega_3) \in \tau_2\}.$$

▶ **Proposition 2.** *For every transduction τ, $\tau \circ \tau_{id} = \tau = \tau_{id} \circ \tau$.*

The *left quotient* $\lceil \cdot, \cdot \rfloor^{-1}$ over transductions is defined as follows: $\forall \omega_1, \omega_2 \in \Gamma^*$ with $|\omega_1| = |\omega_2|$, $\lceil \omega_1, \omega_2 \rfloor^{-1}\tau = \{(\omega, \omega') \mid (\omega_1\omega, \omega_2\omega') \in \tau\}$.

▶ **Proposition 3.** *[30] For every $\omega_1, \omega_2 \in \Gamma^n$, for every transduction τ_1, τ_2,*

$$\lceil \omega_1, \omega_2 \rfloor^{-1}(\tau_1 \circ \tau_2) = \bigcup_{\omega_3 \in \Gamma^{|\omega_1|}} \left((\lceil \omega_1, \omega_3 \rfloor^{-1}\tau_1) \circ (\lceil \omega_3, \omega_2 \rfloor^{-1}\tau_2)\right).$$

Let $\mathcal{T}$ be a set of transductions, the closure $\langle \mathcal{T} \rangle^\cup$ of $\mathcal{T}$ over the composition $\circ$, left quotient $\lceil \cdot, \cdot \rfloor^{-1}$ and union $\cup$ is defined as follows:

- $\mathcal{T} \subseteq \langle \mathcal{T} \rangle^\cup$, $\emptyset \in \langle \mathcal{T} \rangle^\cup$ and $\tau_{id} \in \langle \mathcal{T} \rangle^\cup$;
- if $\tau_1, \tau_2 \in \langle \mathcal{T} \rangle^\cup$, then $\tau_1 \circ \tau_2 \in \langle \mathcal{T} \rangle^\cup$ and $\tau_1 \cup \tau_2 \in \langle \mathcal{T} \rangle^\cup$;
- if $\tau \in \langle \mathcal{T} \rangle^\cup$, then $\lceil \gamma, \gamma' \rfloor^{-1}\tau \in \langle \mathcal{T} \rangle^\cup$ for all $\gamma, \gamma' \in \Gamma$.

Similarly, let $\langle \mathcal{T} \rangle$ denote the closure of $\mathcal{T}$ over the composition $\circ$ and left quotient $\lceil \cdot, \cdot \rfloor^{-1}$.

▶ **Proposition 4.** *(a) The set $\langle \mathcal{T} \rangle$ is finite iff the set $\langle \mathcal{T} \rangle^\cup$ is finite.*

(b) The set $\langle \mathcal{T} \rangle^\cup$ is the semigroup generated by $(\langle \mathcal{T} \rangle, \cup)$, that is, $\forall \tau \in \langle \mathcal{T} \rangle^\cup$, $\exists \tau_1, ..., \tau_m \in \langle \mathcal{T} \rangle$ for $m \geq 1$ such that $\tau = \bigcup_{i=1}^m \tau_i$.

2.2 Pushdown Systems with Transductions

Pushdown systems with transductions (TrPDSs) [30] are an extension of pushdown systems by associating each transition with a transduction which modifies the stack content by applying the transduction. This extension allows TrPDSs to model sequential programs that manipulate the stack content rather than only the top of the stack.

▶ **Definition 5.** A *pushdown system with transductions* (TrPDS) $\mathcal{P}$ is a tuple $(P, \Gamma, \mathcal{T}, \Delta)$, where P is a finite set of control states, Γ is a finite alphabet, $\mathcal{T}$ is a finite set of transductions over Γ^*, $\Delta \subseteq P \times \Gamma \times \mathcal{T} \times P \times \Gamma^*$ is a finite set of transition rules. A TrPDS is a *pushdown system* (PDS) if $\mathcal{T} = \{\tau_{id}\}$.

We will write $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$ instead, if $(p, \gamma, \tau, p', \omega) \in \Delta$. A *configuration* of a TrPDS $\mathcal{P}$ is a pair $\langle p, \omega\rangle \in P \times \Gamma^*$ where p is the control state and ω is the stack content. Let $\mathcal{C}_\mathcal{P}$ denote the set of all the configurations $P \times \Gamma^*$ of the TrPDS $\mathcal{P}$. The TrPDS $\mathcal{P}$ is called *finite* if the set $\langle\mathcal{T}\rangle$ (i.e.,$\langle\mathcal{T}\rangle^\cup$) is finite. If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$, then for every $\omega' \in \Gamma^*$, the configuration $\langle p, \gamma\omega'\rangle$ is an *immediate predecessor* of the configuration $\langle p', \omega u\rangle$ for every $u \in \tau(\omega')$, and the configuration $\langle p', \omega u\rangle$ for every $u \in \tau(\omega')$ is an *immediate successor* of the configuration $\langle p, \gamma\omega'\rangle$. Let $\Longrightarrow \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ be the *immediate successor relation*, i.e., for every $\omega', u \in \Gamma^*$, $\langle p, \gamma\omega'\rangle \Longrightarrow \langle p', \omega u\rangle$ if $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$ and $u \in \tau(\omega')$. A *run* of $\mathcal{P}$ is a sequence of configurations $c_1 c_2 \cdots$ such that for every $i \geq 1$, c_{i+1} is an immediate successor of c_i.

Let $\Longrightarrow^n \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ be the successor relation over configurations of $\mathcal{P}$ defined as follows:

- $c \Longrightarrow^0 c$ for every $c \in \mathcal{C}_\mathcal{P}$;
- $c \Longrightarrow^n c''$ if there exists $c' \in \mathcal{C}_\mathcal{P}$ such that $c \Longrightarrow c'$ and $c' \Longrightarrow^{n-1} c''$.

Let $\Longrightarrow^* \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ denote the reflexive transitive closure of the immediate successor relation $\Longrightarrow$, i.e., $\Longrightarrow^* = \bigcup_{i\geq 0} \Longrightarrow^i$. Let $\Longrightarrow^+ \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ denote the transitive closure of the immediate successor relation $\Longrightarrow$, i.e., $\Longrightarrow^+ = \bigcup_{i\geq 1} \Longrightarrow^i$.

The *predecessor function* $pre : 2^{\mathcal{C}_\mathcal{P}} \longrightarrow 2^{\mathcal{C}_\mathcal{P}}$ of $\mathcal{P}$ is defined as follows: $pre(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c \Longrightarrow c'\}$. The reflexive transitive closure of pre is denoted by pre^*. Formally, $pre^*(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c \Longrightarrow^* c'\}$. Similarly, the *successor function* $post : 2^{\mathcal{C}_\mathcal{P}} \longrightarrow 2^{\mathcal{C}_\mathcal{P}}$ of $\mathcal{P}$ is defined as follows: $post(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c' \Longrightarrow c\}$. The reflexive transitive closure $post^*$ of $post$ is defined as $post^*(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c' \Longrightarrow^* c\}$.

2.3 Finite Automata with Transductions

To finitely represent regular sets of configurations of TrPDSs, we use finite automata with transductions.

▶ **Definition 6** ([30])**.** Given a TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$, a *finite automaton with transduction and ϵ-moves* (ϵ-TrNFA) $\mathbf{A}$ is a tuple $(S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, where S is a finite set of states, $\Lambda \subseteq S \times (\Gamma \cup \{\epsilon\}) \times \langle\mathcal{T}\rangle^\cup \times S$ is a finite set of transition rules, $S_0, S_f \subseteq S$ are initial and final states. An ϵ-TrNFA $\mathbf{A}$ is TrNFA if $\Lambda \subseteq S \times \Gamma \times \langle\mathcal{T}\rangle^\cup \times S$.

We write $s \overset{\gamma|\tau}{\longmapsto} s'$ if $(s, \gamma, \tau, s') \in \Lambda$ (note that $\gamma \in \Gamma \cup \{\epsilon\}$). Let $\longmapsto^n : S \times \Gamma^* \times \langle\mathcal{T}\rangle^\cup \times S$ be a relation over states of $\mathbf{A}$ defined as follows:

- $s \overset{\epsilon|\tau_{id}}{\longmapsto} {}^0 s$, for every $s \in S$;
- $s \overset{\gamma\gamma_1\cdots\gamma_n|\left(\lceil\gamma_1\cdots\gamma_n,\gamma'_1\cdots\gamma'_n\rfloor^{-1}\tau\right)\circ\tau_1}{\longmapsto} {}^{n+1}\, s_2$ for all $\gamma_1, ..., \gamma_n \in \Gamma$, if $\exists s_1 \in S$ such that $s \overset{\gamma|\tau}{\longmapsto} s_1$ and $s_1 \overset{\gamma'_1\cdots\gamma'_n|\tau_1}{\longmapsto} {}^n s_2$.

TrNFA is the standard finite state automata if $\mathcal{T} = \{\tau_{id}\}$, a.k.a. $\mathcal{P}$-automata [6] if S corresponds to the control states of $\mathcal{P}$.

Let $\longmapsto^* = \bigcup_{i\geq 0} \longmapsto^i$. A configuration $\langle p, \omega\rangle \in P \times \Gamma^*$ of a TrPDS $\mathcal{P}$ is *recognized* (accepted) by an ϵ-TrNFA $\mathbf{A}$ iff $s \overset{\omega|\tau}{\longmapsto} {}^* s'$ such that $s = p \in S_0$, $s' \in S_f$ and $(\epsilon, \epsilon) \in \tau$. A set C of configurations is *rational* (regular) if there exists an ϵ-TrPDS $\mathbf{A}$ such that $L(\mathbf{A}) = C$. From now on, we omit the paths of the form $s_1 \overset{\omega|\tau}{\longmapsto} {}^n s_2$ such that $\tau = \emptyset$, as these paths do not allow the ϵ-TrNFA to accept a configuration.

▶ **Theorem 7** ([30]). *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$, both $post^*(C)$ and $pre^*(C)$ are rational and effectively computable.*

3 Computing pre^*

In this section, we present a saturation procedure to compute pre^* which is different from the way presented in [30] and an efficient implementation for pre^*.

3.1 Saturation Procedure for Computing pre^*

Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ that recognizes a rational set of configurations of $\mathcal{P}$, w.l.o.g., we assume $P = S_0$ and there is no transition rule in $\mathbf{A}$ leading to an initial state and $\mathbf{A}$ uses only the identity transduction τ_{id} (cf. Section 6.4 of [30]), we construct a new TrNFA $\mathbf{A}^{pre^*} = (S, \Gamma, \Lambda^{pre^*}, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}^{pre^*}$ recognizes $pre^*(L(\mathbf{A}))$, i.e., $L(\mathbf{A}^{pre^*}) = pre^*(L(\mathbf{A}))$. The construction of $\mathbf{A}^{pre^*}$ is based on a kind of saturation procedure which extends the saturation procedure to compute pre^* of PDSs [6]. Initially, $\mathbf{A}^{pre^*} = \mathbf{A}$, then we iteratively apply the following saturation procedure until no new transition rule can be added into $\mathbf{A}^{pre^*}$.

> If $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle q, \omega\rangle \in \Delta$ and $q \overset{\omega|\tau_2}{\longmapsto} {}^* q'$ in the current automaton $\mathbf{A}^{pre^*}$, add a transition rule $p \overset{\gamma|\tau_1\circ\tau_2}{\longmapsto} q'$ into Λ^{pre^*}.

Since the set of states of $\mathbf{A}^{pre^*}$ and the set $\langle\mathcal{T}\rangle$ of transductions are finite, the set of transition rules in $\mathbf{A}^{pre^*}$ is finite. Thus, the above saturation will eventually reach a fixpoint. Intuitively, if there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle q, \gamma_1^1 \cdots \gamma_n^1\rangle \in \Delta$, then $\langle p, \gamma\gamma_{n+1}\cdots\gamma_m\rangle \Longrightarrow \langle q, \gamma_1^1\cdots\gamma_n^1\gamma_{n+1}^1\cdots\gamma_m^1\rangle$ for all $\gamma_{n+1}^1\cdots\gamma_m^1 \in \tau(\gamma_{n+1}\cdots\gamma_m)$. If the automaton $\mathbf{A}^{pre^*}$ recognizes the configuration $\langle q, \gamma_1^1\cdots\gamma_m^1\rangle$ by a path $q \overset{\gamma_1^1\cdots\gamma_m^1|\tau'}{\longmapsto}{}^m g$ for some final state g of $\mathbf{A}^{pre^*}$ and $(\epsilon, \epsilon) \in \tau'$, then, we can decompose this path to $q \overset{\gamma_1^1\cdots\gamma_n^1|\tau''}{\longmapsto}{}^n q'$ and $q' \overset{\gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}|\tau'''}{\longmapsto}{}^{m-n} g$ such that if $\tau' = (\lceil\gamma_2^1\cdots\gamma_m^1, \gamma_2^2\cdots\gamma_m^2\rfloor^{-1}\tau_1)\circ\cdots\circ(\lceil\gamma_m^{m-1}, \gamma_m^m\rfloor^{-1}\tau_{m-1})\circ\tau_m$, then

- $\tau'' = (\lceil\gamma_2^1\cdots\gamma_n^1, \gamma_2^2\cdots\gamma_n^2\rfloor^{-1}\tau_1)\circ\cdots\circ(\lceil\gamma_n^{n-1}, \gamma_n^n\rfloor^{-1}\tau_{n-1})\circ\tau_n$,
- $\tau''' = (\lceil\gamma_{n+2}^{n+1}\cdots\gamma_m^{n+1}, \gamma_{n+2}^{n+2}\cdots\gamma_m^{n+2}\rfloor^{-1}\tau_{n+1})\circ\cdots\circ(\lceil\gamma_m^{m-1}, \gamma_m^m\rfloor^{-1}\tau_{m-1})\circ\tau_m$.

Moreover, since $(\epsilon, \epsilon) \in \tau'$, we get that $(\gamma_{n+1}^1\cdots\gamma_m^1, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}) \in \tau''$ and $(\epsilon, \epsilon) \in \tau'''$. Applying the saturation procedure, the transition rule $p \overset{\gamma|\tau\circ\tau''}{\longmapsto} q'$ is added into $\mathbf{A}^{pre^*}$. Therefore, $\mathbf{A}^{pre^*}$ recognizes the configuration $\langle p, \gamma\gamma_{n+1}\cdots\gamma_m\rangle$ by composing $p \overset{\gamma|\tau\circ\tau''}{\longmapsto} q'$ and $q' \overset{\gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}|\tau'''}{\longmapsto}{}^{m-n} g$ into $p \overset{\gamma\gamma_{n+1}\cdots\gamma_m|\left(\lceil\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}\rfloor^{-1}(\tau\circ\tau'')\right)\circ\tau'''}{\longmapsto}{}^{m-n+1} g$ (note that $(\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}) \in \tau\circ\tau''$ implies that $(\epsilon, \epsilon) \in \left(\lceil\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}\rfloor^{-1}(\tau\circ\tau'')\right)\circ\tau'''$. Thus, we get the following theorem.

▶ **Theorem 8.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$ recognized a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can construct a TrNFA $\boldsymbol{A}'$ such that $L(\boldsymbol{A}') = pre^*(C)$ in time $\mathbf{O}(|\Delta|^3\cdot|S|^3\cdot|\Lambda|\cdot f(|\mathcal{T}|))$ and in space $\mathbf{O}(|\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$, where f is some computable function.*

We notice that the number $|\Lambda^{pre^*}|$ of transition rules of $\mathbf{A}^{pre^*}$ is at most $\mathbf{O}(|\Lambda| + |\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$. For each transition rule $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle q, \gamma_1\gamma_2\rangle \in \Delta$, paths $q \overset{\gamma_1\gamma_2|\tau_2}{\longmapsto}{}^* g$ can be computed in time $\mathbf{O}(f(|\mathcal{T}|)\cdot(|S| + |P|)\cdot|\Lambda^{pre^*}|)$ for some computable function f. Thus, we get that the saturation procedure executes at most in time $\mathbf{O}(|\Delta|^3\cdot|S|^3\cdot|\Lambda|\cdot f(|\mathcal{T}|))$. Memory is needed for storing the new transition rules which is bounded by $\mathbf{O}(|\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$.

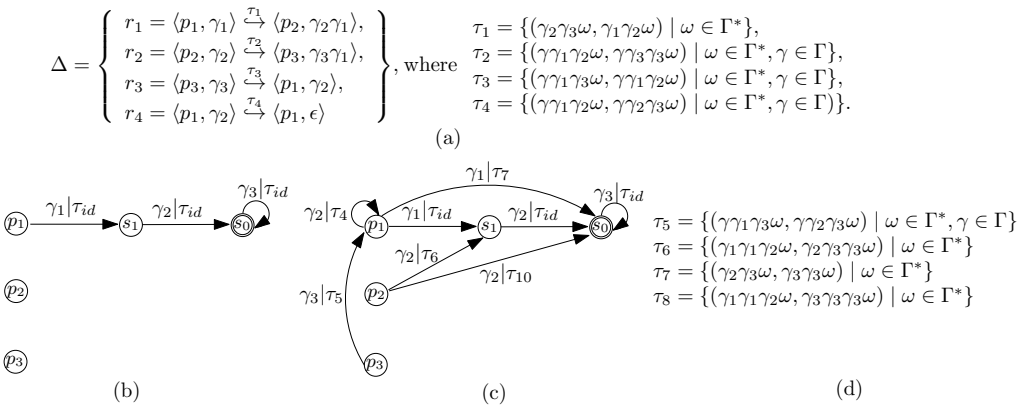


Figure 1 (a) The set of transition rules Δ, (b) the TrNFA $\mathbf{A}$, (c) the TrNFA $\mathbf{A}^{pre^*}$ and (d) consists of related transductions.

▶ Remark. In [30], the authors introduce TrNFA and present a saturation procedure to compute pre^* without its complexity. They define the relation $\longmapsto^*$ by introducing a *pseudo formal power series semiring* to solve the associativity problem of the composition of transitions of TrNFAs. Their saturation procedure is proceeded based on this semiring. Our approach is proceeded based on TrNFAs and we show that this problem is fixed-parameter tractable (FPT). We believe our direct approach is more convenient for studying optimal algorithm or BDD-based symbolic techniques.

▶ **Example 9.** Consider the TrPDS with control states $\{p_1, p_2, p_3\}$ and Δ as shown in Figure 1(a). Let $\mathbf{A}$ be the TrNFA as shown in Figure 1(b). The result of applying the saturation procedure is shown in Figure 1(c). .

3.2 An Efficient Algorithm for Computing pre^*

In this section, we present an efficient implementation of the saturation procedure given in Section 3.1. W.l.o.g., we suppose in this section that for every TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$, $|\omega| \leq 2$ for every transition rule $\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle q, \omega \rangle \in \Delta$. Let Δ_i denote $\{\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g, \omega \rangle \in \Delta \mid |\omega| = i\}$, for every $i \in \{0, 1, 2\}$.

Algorithm 1 computes the transition rules of $\mathbf{A}^{pre^*}$ by implementing the saturation procedure from Section 3.1. The basic idea follows from the efficient algorithm for computing pre^* of PDSs [13] which avoids unnecessary operations. Intuitively, for the transition rules of the form $\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon \rangle$ or $\langle p, \gamma_1 \rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma \rangle$ in Δ, the algorithm proceeds exactly the same as the saturation procedure given in Section 3.1. Whenever $\mathcal{P}$ has a transition rule in the form of $\langle p, \gamma_1 \rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\gamma_2 \rangle$, we look out for every $q', q'' \in S$ and $\gamma_2' \in \Gamma$, the pairs of transition rules $q \overset{\gamma|\tau}{\longmapsto} q'$ and $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ such that $\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau \neq \emptyset$, so that we can add the transition rule $p \overset{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau) \circ \tau_2}{\longmapsto} q''$. However, the order of such transitions added into the automaton $\mathbf{A}^{pre^*}$ can be arbitrary. Whenever a transition rule like $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ is found, we have to check whether $q \overset{\gamma|\tau}{\longmapsto} q'$ exists or not. Then, this checking may be negative, and wastes time to no avail. However, once a transition rule $q \overset{\gamma|\tau}{\longmapsto} q'$ is seen, we know that all subsequent transitions like $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ must lead to the addition of the transition rule $p \overset{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau) \circ \tau_2}{\longmapsto} q'$. That's why we introduce a new transition rule $\langle p, \gamma_1 \rangle \overset{\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau)}{\hookrightarrow} \langle q', \gamma_2' \rangle$ into Δ' which allows

Input : A finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}$ uses only τ_{id} and Λ has no transition rule leading to a state in P

Output : The set of transition rules of $\mathbf{A}^{pre^*}$

1 $\Lambda' := \Lambda$; $trans := \Lambda$; $\Delta' := \emptyset$;
2 **foreach** $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$ **do** **Update**$(p \overset{\gamma|\tau}{\longmapsto} p')$;
3 ;
4 **while** $trans \neq \emptyset$ **do**
5 remove $t = q \overset{\gamma|\tau}{\longmapsto} q'$ from $trans$;
6 **foreach** $\langle p, \gamma_1\rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\rangle \in \Delta \cup \Delta'$ **do** **Update**$(p \overset{\gamma_1|\tau' \circ \tau}{\longmapsto} q')$;
7 ;
8 **foreach** $\langle p, \gamma_1\rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\gamma_2\rangle \in \Delta$ *and* $\gamma_2' \in \Gamma$ **do**
9 **if** $\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau \neq \emptyset$ **then**
10 $\Delta' := \Delta' \cup \{\langle p, \gamma_1\rangle \xrightarrow{\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau)} \langle q', \gamma_2'\rangle\}$;
11 **foreach** $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q'' \in \Lambda'$ **do**
12 **Update**$(p \xrightarrow{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau) \circ \tau_2} q'')$;
13 **return** Λ';
14 **Procedure Update**$(q \overset{\gamma|\tau}{\longmapsto} q')$
15 **if** $t = q \overset{\gamma|\tau'}{\longmapsto} q' \in \Lambda'$ **then**
16 $t' := q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$;
17 $\Lambda' := \Lambda' \cup \{t'\} \setminus \{t\}$;
18 **if** $\tau' \neq \tau' \cup \tau$ **then** $trans := trans \cup \{t'\} \setminus \{t\}$;
19 ;
20 **else if** $\tau \neq \emptyset$ **then**
21 $\Lambda' := \Lambda' \cup \{q \overset{\gamma|\tau}{\longmapsto} q'\}$;
22 $tran := tran \cup \{q \overset{\gamma|\tau}{\longmapsto} q''\}$;

Algorithm 1. An efficient algorithm for computing pre^*.

us to add the transition rule $p \xrightarrow{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau) \circ \tau_2} q''$ once $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ occurs. Let us explain Algorithm 1 line by line as follows.

Line 1 initializes the algorithm by assigning Λ to Λ' and $trans$, $\emptyset$ to Δ'. Line 2 handles normal transition rules of the form $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle$, where new transitions $p \overset{\gamma|\tau}{\longmapsto} p'$ can be immediately added. Once a new transition rule is created, we call the procedure **Update** which will be explained later. Lines 3-10 iteratively removes a transition $t = q \overset{\gamma|\tau}{\longmapsto} q'$ from $trans$ until it is empty. The loop at Line 5 handles the case when q and γ match the right-hand side of transition rules in $\Delta \cup \Delta'$.

The procedure **Update** listed at Lines 12-19 is called whenever a new transition rule $q \overset{\gamma|\tau}{\longmapsto} q'$ is created. If Λ' contains a transition rule of the form $t = q \overset{\gamma|\tau'}{\longmapsto} q'$ for any τ', then, we remove t from Λ' and add a new transition rule $q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$ into Λ' at Line 15. In other words, we update the transduction τ' by $\tau' \cup \tau$. Moreover, if $\tau' \cup \tau$ does not equal to τ', we remove t from $trans$ and add $q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$ into $trans$ at Line 16 for later processing. Otherwise if Λ' has no transition rule like t, we add $q \overset{\gamma|\tau}{\longmapsto} q'$ into Λ' and $trans$.

▶ **Theorem 10.** *Given a finite TrPDS* $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ *and a TrNFA* $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, *we can compute a TrNFA* $\boldsymbol{A}^{pre^*}$ *in time* $\mathbf{O}(|S|^2 \cdot f(|\mathcal{T}|) \cdot |\Delta| \cdot |\Gamma|)$ *for some computable function* f *and in space* $\mathbf{O}(|S| \cdot |\Delta| \cdot |\langle\mathcal{T}\rangle| \cdot |\Gamma|)$ *such that* $L(\boldsymbol{A}^{pre^*}) = pre^*(L(\boldsymbol{A}))$.

4 Computing $post^*$

In this section, we present an approach to compute $post^*$ which is different from the way presented in [30]. In [30], $post^*$ is computed by transforming a *finite* TrPDS into an equivalent PDS and then computing $post^*$ of the resulting PDS. We will present a saturation procedure which directly computes $post^*$ similar as computing pre^* given in Section 3. Finally, we give an efficient algorithm implementing this saturation procedure.

4.1 Saturation Procedure for Computing $post^*$

Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ that recognizes a rational set of configurations of $\mathcal{P}$, we can construct an ϵ-TrNFA $\mathbf{A}^{post^*} = (S^{post^*}, \Lambda^{post^*}, S_0, S_f)$ such that $\mathbf{A}^{post^*}$ recognizes $post^*(L(\mathbf{A}))$, i.e., $L(\mathbf{A}^{post^*}) = post^*(L(\mathbf{A}))$. W.l.o.g., we assume that $S_0 = P$ and there is no transition rule in $\mathbf{A}$ leading to an initial state and $\mathbf{A}$ uses only the identity transduction τ_{id}. The construction of $\mathbf{A}^{post^*}$ is similar than the construction of $\mathbf{A}^{pre^*}$ which is an extension of the saturation procedure for computing $post^*$ of PDSs [13].

Given a transduction τ, let $\overline{\tau}$ denote the inversion $\{(\omega_1, \omega_2) \mid (\omega_2, \omega_1) \in \tau\}$ of τ, let $\overline{T}$ denote $\bigcup_{\tau \in T} \overline{\tau}$ for a given set T of transductions.

▶ **Proposition 11.** $\overline{\langle\mathcal{T}\rangle} = \langle\overline{\mathcal{T}}\rangle$ *and* $\overline{\langle\mathcal{T}\rangle^\cup} = \langle\overline{\mathcal{T}}\rangle^\cup$.

Initially, $\mathbf{A}^{post^*} = \mathbf{A}$, then we iteratively apply the following saturation procedure until the automaton is *saturated* (i.e., no new transition rule can be added):

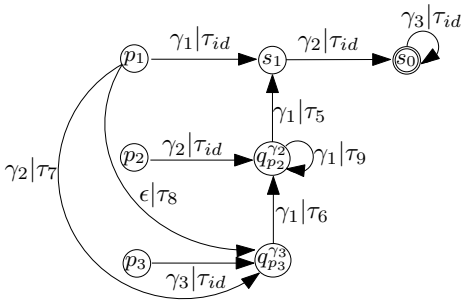
(i) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\epsilon|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*};

(ii) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\gamma_1|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*};

(iii) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\gamma_2\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\gamma_1|\ \tau_{id}}{\longmapsto} q_{p'}^{\gamma_1}$ and $q_{p'}^{\gamma_1} \overset{\gamma_2|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*} and add a new state $q_{p'}^{\gamma_1}$ into S^{post^*}.

Intuitively, if there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rule $p' \overset{\epsilon|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.

If there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \gamma_1\omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \gamma_1\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rule $p' \overset{\gamma_1|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \gamma_1\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.



$\tau_5 = \{(\gamma_1\gamma_2\omega, \gamma_2\gamma_3\omega) \mid \omega \in \Gamma^*\}$
$\tau_6 = \{(\gamma\gamma_3\gamma_3\omega, \gamma\gamma_1\gamma_2\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_7 = \{(\gamma\gamma_1\gamma_2\omega, \gamma\gamma_1\gamma_3\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_8 = \{((\gamma\gamma_2\gamma_3\omega, \gamma\gamma_1\gamma_3\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_9 = \{(\gamma_1\gamma_2\gamma_3\omega, \gamma_1\gamma_1\gamma_2\omega) \mid \omega \in \Gamma^*\}$

Figure 2 The resulting TrNFA $\mathbf{A}^{post^*}$.

If there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\gamma_2\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \gamma_1\gamma_2\omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \gamma_1\gamma_2\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rules $p' \overset{\gamma_1|\ \tau_{id}}{\longmapsto} q_{p'}^{\gamma_1}$ and $q_{p'}^{\gamma_1} \overset{\gamma_2|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \gamma_1\gamma_2\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.

▶ **Theorem 12.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$ recognized a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can construct a TrNFA $\boldsymbol{A}'$ such that $L(\boldsymbol{A}') = post^*(C)$.*

▶ **Example 13.** Consider the TrPDS shown in Figure 1(a) and the TrNFA $\mathbf{A}$ shown in Figure 1(b). The result of applying the saturation procedure is shown in Figure 2.

4.2 An Efficient Algorithm for Computing $post^*$

In this section, we present an efficient implementation of the saturation procedure given in Section 4.1 which avoids unnecessary operations. Given a rational set of configurations of C represented by a TrNFA $\mathbf{A}$.

Algorithm 2 computes the transition rules of $\mathbf{A}^{post^*}$ by implementing the saturation procedure given in Section 4.1. The approach is similar to the solution for efficiently computing pre^*. We use *trans* to store the transition rules that we still need to examine. Lines 1-2 initialize the algorithm. Initially, Λ' is equal to Λ, while *trans* is equal to $\Lambda \cap P \times \Gamma \times \mathcal{T} \times S$, as transition rules starting from states outside of P do not need to be examined. The set S^{post^*} of states is equal to $S \cup \{q_{p_1}^{\gamma_1} \mid \langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta\}$ as described in Section 4.1. The algorithm iteratively removes a transition $t = p \overset{\gamma|\tau}{\longmapsto} q$ from *trans* until it is empty. The loops at Line 6, Line 7 and Lines 8-10 handle the case when p and γ match the left-hand sides of transition rules in Δ. This is done similar as the saturation rules (i), (ii), and (iii), respectively. The loops at Lines 11-12 and Lines 13-14 handle ϵ-transition rules. In the saturation procedure given in Section 4.1, we have to compute paths $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ which may involve several ϵ-transitions. In Algorithm 2, we solve this problem by combining transition pairs of the form $p \overset{\epsilon|\tau_1}{\longmapsto} q_1$ and $q_1 \overset{\gamma|\tau_2}{\longmapsto} q$ into transition rules $p \overset{\gamma'|(\lceil\gamma',\gamma\rfloor^{-1}\tau_1)\circ\tau_2}{\longmapsto} q$ for $\gamma' \in \Gamma$ whenever such a pair is found.

Input : A finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}$ uses only τ_{id}, Λ has no transition leading to a state in P and no ϵ-transition

Output : The TrNFA $\mathbf{A}^{post^*} = (S^{post^*}, \Lambda', S_0, S_f)$

$\Lambda' := \Lambda$; $trans := \Lambda \cap P \times \Gamma \times \mathcal{T} \times S$;
$S^{post^*} := S \cup \{q_{p_1}^{\gamma_1} \mid \langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta\}$;
while $trans \neq \emptyset$ **do**
  remove $t = p \overset{\gamma|\tau}{\longmapsto} q$ from $trans$;
  if $\gamma \neq \epsilon$ **then**
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \epsilon\rangle \in \Delta$ **do** **Update**$(p_1 \overset{\epsilon \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
    ;
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \gamma_1\rangle \in \Delta$ **do** **Update**$(p_1 \overset{\gamma_1 \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
    ;
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta$ **do**
      Update$(p_1 \overset{\gamma_1 \mid \tau_{id}}{\longmapsto} q_{p_1}^{\gamma_1})$;
      Update$(q_{p_1}^{\gamma_1} \overset{\gamma_2 \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
      foreach $p_2 \overset{\epsilon \mid \tau_2}{\longmapsto} q_{p_1}^{\gamma_1} \in \Lambda',\ \gamma_2' \in \Gamma$ **do**
        Update$(p_2 \overset{\gamma_2' \mid (\lceil\gamma_2', \gamma_2\rfloor^{-1}\tau_2)\circ\overline{\tau_1}\circ\tau}{\longmapsto} q)$
  else **foreach** $q \overset{\gamma_1 \mid \tau_1}{\longmapsto} q' \in \Lambda',\ \gamma_1' \in \Gamma$ **do**
    Update$(p \overset{\gamma_1' \mid (\lceil\gamma_1', \gamma_1\rfloor^{-1}\tau)\circ\tau_1}{\longmapsto} q')$
  ;
return $\mathbf{A}^{post^*}$;

Algorithm 2. An efficient algorithm for computing $post^*$.

▶ **Theorem 14.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can compute a TrNFA $\boldsymbol{A}^{post^*}$ in $\mathbf{O}(|S|\cdot f(|\mathcal{T}|)\cdot|\Delta|^3\cdot|\Gamma|)$ time and space for some computable function f such that $L(\boldsymbol{A}^{post^*}) = post^*(L(\boldsymbol{A}))$.*

5 Application

In [30], Uezato and Minamide presented two potential applications of TrPDSs: checking reachability of conditional PDSs [22, 15] and discrete-timed PDSs [1] via pre^* or $post^*$ computing of TrPDSs. In this section, we will present another potential application of TrPDSs. We show how the presence of transductions enables the modeling of Boolean programs with call-by-reference parameter passing. Boolean programs in which all variables and parameters (call-by-value) have Boolean type are thought of as an abstract representation of C/C++ programs with recursion [3]. In their definition, Boolean programs contain procedures with call-by-value parameter passing rather than call-by-reference parameter passing. While call-by-reference parameter passing is a widely used programming paradigm in C/C++, Java, etc. Using TrPDSs, we can verify safety properties of Boolean programs with call-by-reference parameter passing.

5.1 Boolean Programs with Call-by-Reference Parameter Passing

A *Boolean program* BP is a tuple $(Proc, main, G)$, where $Proc$ is a finite set of procedures, $main \in Proc$ is the *initial* procedure, G is a finite set of global Boolean variables. Every procedure $r \in Proc$ is a tuple (N_r, E_r, L_r), where N_r is a finite set of control points with r_{entry} as the unique entry node, E_r is a finite set of edges, L_r is the finite set of local Boolean variables in r. W.l.o.g., we assume that $L_r \cap G = \emptyset$ for all $r \in Proc$. $L'_r = L_r \cup G$ is a set of *visible variables* in r. Let L_r^{ref} be the set of all the call-by-reference formal parameters of the procedure r, L_n^{ref} (resp. L_n) be the set L_r^{ref} (resp. L_r) such that $n \in N_r$. Given a procedure call $stmt = r(v_1, ..., v_m)$ at the control point n whose return address is n', for every formal parameter v' of r, let $fRef(n')(v') \in \{v_1, ..., v_m\}$ be the actual parameter of v' at the caller site n'.

A *valuation* $\xi \subseteq L'_r$ is a subset of L'_r meaning that the Boolean value of $x \in L'_r$ is 1 if $x \in \xi$, otherwise 0. Let $\xi(x) = 1$ if $x \in \xi$, otherwise 0. Let $\xi[d/x]$ be the valuation such that $\xi[d/x](y) = d$ if $x = y$, otherwise $\xi(y)$. The edges in E_r are of the form $(n, \xi, stmt, n')$ meaning that n' is the next control point of n when the valuation at n is ξ, where $stmt$ is the statement at n. Let $[\![stmt]\!]_\xi$ be the valuation after executing the statement $stmt$ that is neither a procedure call nor return. The details of the execution model and semantics of other statements refer to [3].

5.2 Modeling Approach

W.l.o.g., we assume that call-by-reference actual parameters are local variables. Indeed, global variables are always visible for all procedures and do not need to be passed by parameters. Different from call-by-value parameter passing which keeps its own copy at callee site, a parameter passed by call-by-reference will not keep its own copy at callee site. Precisely speaking, the values of call-by-reference actual parameters at a caller procedure are always same as the values of the corresponding formal parameters at the callee procedure. We will use transductions to encode the changing of call-by-reference formal parameters, as transductions in TrPDSs allow us to manipulate the stack content rather than the top of stack in PDSs.

The construction of TrPDSs from Boolean programs BP with call-by-reference parameter passing follows the standard modeling approach of PDSs from Boolean programs [13] except the assignments with call-by-reference formal parameter as left value for which the side-effect of assignments should also effect on the value of the corresponding actual parameters at the corresponding caller site. The valuations of global variables G are put in the control locations of the TrPDSs, the pairs of the valuations of local variables and control points (i.e., nodes) of the program are stored in the stack of the TrPDSs. The TrPDS model will be $\mathcal{P} = (2^G, \bigcup_{r \in Proc}(N_r \times 2^L_r), \mathcal{T}, \Delta)$. A configuration of the TrPDS model is in the form of $c = \langle \xi, (n_0, \xi_0) \cdots (n_k, \xi_k) \rangle$ meaning that the execution of BP is at the control point n_0 with ξ as the valuation of global variables, ξ_0 as the valuation of local variables of the procedure containing n_0. Moreover, $(n_1, \xi_1) \cdots (n_k, \xi_k)$ is the calling history of the execution such that for every $i : 1 \leq i \leq k$, n_i is the return address of the procedure call that jump into the procedure containing n_{i-1} and ξ_i is the stored valuation of local variables when the procedure call is made. Different from Boolean programs only with call-by-value parameter passing, a local variable of a procedure may be a call-by-reference parameter of the procedure. In this case, the value of a local variable and its referenced variable are identical. Therefore, the potential possible configurations of the TrPDS model should only have admitted valuations with respect to the local variables and their referenced variables.

Formally, a word $(n_0,\xi_0)\cdots(n_k,\xi_k)$ or a configuration $\langle \xi,(n_0,\xi_0)\cdots(n_k,\xi_k)\rangle$ is *admissible* if for every $i: 0 \le i \le k-1$ and every $v \in L^{ref}_{n_i}$, $v \in \xi_i$ iff $fRef(n_{i+1})(v) \in \xi_{i+1}$.

Given two variable sets $\xi'_0,\xi_0 \subseteq \bigcup_{r\in Proc} L^{ref}_r$, let $\overrightarrow{(\xi'_0,\xi_0)} \subseteq \Gamma^*\times\Gamma^*$ be the transduction such that for every $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \Gamma^*\times\Gamma^*$, $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \overrightarrow{(\xi'_0,\xi_0)}$ iff $(n_1,\xi_1)...(n_k,\xi_k)$ is admissible, and for every $i: 1 \le i \le k$, $\xi'_i = \xi_i \cup \{fRef(n_i)(v) \mid v \in \xi'_{i-1} \setminus \xi_{i-1}\} \setminus \{fRef(n_i)(v) \mid v \in \xi_{i-1} \setminus \xi'_{i-1}\}$. Intuitively, given an admissible word $(n_1,\xi_1)...(n_k,\xi_k)$ and two sets ξ_0 and ξ'_0 denoting respectively the valuations of local variables before and after an assignment, $(n_1,\xi'_1)...(n_k,\xi'_k)$ is the admissible word obtained from $(n_1,\xi_1)...(n_k,\xi_k)$ with the updating of all the actual parameters of the corresponding formal call-by-reference parameters with respect to ξ'_0 and ξ_0.

The set Δ of transition rules that mimic the control flow of BP is defined as follows: for every edge $e=(n,\xi,stmt,n')$ in a procedure r,

- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_{id}}{\hookrightarrow} \langle \xi\cap G,(r'_{enrty},\xi')(n',\xi\cap L_r)\rangle \in \Delta$ if $stmt$ is a procedure call $r'(v_1,...,v_m)$, where $\xi' = \{v \in L_{r'} \mid fRef(n')(v) \in \xi\}$;
- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_{id}}{\hookrightarrow} \langle \xi\cap G,\epsilon\rangle \in \Delta$ if $stmt$ is a return,
- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_e}{\hookrightarrow} \langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)\rangle \in \Delta$ otherwise, where $\tau_e = \overrightarrow{([\![stmt]\!]_\xi,\xi)}$.

The set $\mathcal{T}$ of transductions is $\{\tau_{id},\tau_e \mid e=(n,\xi,stmt,n') \in E_r, r\in Proc, stmt$ is neither a procedure call nor return$\}$. Intuitively, the TrPDS model mimics the execution of BP. The intuition behind function calls and returns is similar as translating from Boolean programs into PDSs. For details refer to [13]. We explain the assignments.

Suppose the execution of BP is at the control point n with the valuation ξ, and the calling history is $(n_1,\xi_1)...(n_k,\xi_k)$. If the edge $e=(n,\xi,stmt,n')$ is in a procedure r such that $stmt$ is neither a procedure call nor return, then, the execution of BP will move from n to the next point n' with the valuation $[\![stmt]\!]_\xi$. Moreover, the local variables at n_1 that corresponds to the formal call-by-reference parameters in r should take the values of the call-by-reference parameters at n'. Similarly, for every $i: 2 \le i \le k$, the local variables at n_i that corresponds to the formal call-by-reference parameters in the procedure containing n_{i-1} should take the values of the call-by-reference parameters at n_i. Therefore, we add the transition rule $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_e}{\hookrightarrow} \langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)\rangle$ into Δ which allows the TrPDS model to move from the configuration $\langle \xi\cap G,(n,\xi\cap L_r)(n_1,\xi_1)...(n_k,\xi_k)\rangle$ to $\langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)(n_1,\xi'_1)...(n_k,\xi'_k)\rangle$ for every $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \tau_e$. The transduction $\tau_e = \overrightarrow{([\![stmt]\!]_\xi,\xi)}$ correctly specifies the updating of all the actual parameters of the corresponding call-by-reference parameters in the calling history.

▶ Remark. From the definitions of transductions $\mathcal{T}$ and admissible, we can see that all the reachable configurations in the TrPDS model from an admissible configuration are also admissible.

Given two transductions $\tau_1 = \overrightarrow{(\xi'_1,\xi_1)}$, $\tau_2 = \overrightarrow{(\xi'_2,\xi_2)} \in \mathcal{T}$, then

$$\tau_1\circ\tau_2 = \begin{cases} \emptyset & \text{if } \xi'_1 \neq \xi_2,\\ \overrightarrow{(\xi'_2,\xi_1)} & \text{otherwise.}\end{cases}$$

Given two symbols $(n_1,\xi_1),(n'_1,\xi'_1) \in \Gamma^*$ and a transduction $\tau = \overrightarrow{(\xi',\xi)} \in \mathcal{T}$, $\lceil (n_1,\xi_1),(n'_1,\xi'_1)\rfloor^{-1}\tau$ is $\overrightarrow{(\xi'_1,\xi_1)}$ if $n_1 = n'_1 \wedge \xi'_1 = (\xi_1 \cup \{fRef(n_1)(v) \mid v \in \xi' \setminus \xi\}) \setminus \{fRef(n_1)(v) \mid v \in \xi \setminus \xi'\}$, $\emptyset$ otherwise.

Then, we can get that $\langle\mathcal{T}\rangle \subseteq \{\overrightarrow{(\xi',\xi)} \mid \exists r \in Proc : \xi,\xi' \subseteq L_r\}$ which is finite.

▶ **Theorem 15.** *The Boolean program BP can reach a control point n of the procedure r with the valuation ξ and the calling history ω from a control point n' of r' with the valuation ξ' and the calling history ω' iff* $\langle \xi' \cap G, (n', \xi' \cap L_{r'})\omega' \rangle \Longrightarrow^* \langle \xi \cap G, (n, \xi \cap L_r)\omega \rangle$.

Using Theorem 15, we can verify safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing via solving the reachability problem of TrPDSs. The efficiency heavily relies upon the number of transductions from the modeling of the Boolean program and the saturation. From the modeling, the number of transductions is linear in the size of the edges labeled by assignments which assign values to reference variables. During the saturation procedure, transductions are computing via left quotient and composition operators. Therefore, the number of transductions added by the saturation procedure is exponential in the size of return nodes in the Boolean program and doubly exponential in the size of reference variables.

▶ Remark. One may argue that Boolean program with mixed call-by-reference and call-by-value parameter passing can be translated into a Boolean program with only call-by-value parameter passing by using global variables which can be verified by existing techniques such as [3]. However, this will leads to larger state space and may degrade performance.

6 Related Work

Model-checking techniques for PDSs were widely studied and applied to program analysis in the literature [6, 13, 16, 17, 26]. PDSs with checkpoint were introduced in [15] as an extension of PDSs. PDSs with checkpoint can inspect the stack content and are applied to analyse programs with runtime inspection. The reachability problem and LTL model-checking for PDSs with checkpoint were studied in [15] and were applied to the analysis of the HTML5 parser specification in [22]. CTL model-checking for PDSs with checkpoint was studied in [25, 28]. A similar extension of PDSs was used to formulate abstract garbage collection in the control flow analysis of higher-order programs [12].

Weighted PDSs and extended weighted PDSs were introduced in [23, 19] for data-flow analysis purpose. These two extensions associate transitions with elements from semiring domains. The reachability problem is decidable for bounded idempotent semiring. (Extended) weighted PDSs and TrPDSs are quite different two computation models. At least, the elements from semiring can neither inspect nor modify the stack content except the top most symbol on the stack.

Recently, well-structured PDSs (WSPDSs) that combine well-structured transition systems and PDSs was introduced by [10] in which the infinite set of control states and the infinite stack alphabet are well-quasi-order. WSPDS is a powerful model in which recursive vector addition system with states [4, 5], multi-set PDSs [24] and dense-timed PDSs are subsumed [11]. However, the reachability problem is undecidable for WSPDSs. But coverability becomes decidable when the set of control states is finite. In TrPDSs, the set of control states and the stack alphabet are both finite, but the transductions can inspect and modify the stack content.

We should clarify the relation between our work and the work [30]. TrPDSs were first introduced in [30] and are generalization of PDSs with checkpoint and discrete-timed PDSs. The authors showed that TrPDSs can be simulated by PDSs and proposed a saturation procedure to compute pre^* which different from ours. Indeed, our approach is essential to get an efficient implementation algorithm. We also proposed a saturation procedure to compute $post^*$ and its efficient implementation algorithm. These two efficient implementation algorithms necessarily improve the complexity due to the fact that the algorithms have

better complexity than the saturation procedures for pushdown systems. Moveover, we presented a potential application of TrPDSs to modeling and verifying Boolean programs with call-by-reference parameter passing.

7 Conclusion and Future Work

We introduced two saturation procedures to compute pre^* and $post^*$. We also presented two efficient implementation algorithms for the saturation procedures and measured their complexity. We showed that TrPDSs are powerful enough to model Boolean programs with call-by-reference parameter passing. This allows us to verify safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing.

In future, we plan to implement our techniques in a tool and investigate BDD-based symbolic algorithms by representing transductions and valuations of global and local variables in BDDs.

Acknowledgements. We want to thank Lijun Zhang and Zhilin Wu for discussions and suggestions.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of LICS*, pages 35–44, 2012.
2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *Proceedings of LATA*, pages 58–69, 2012.
3 Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *Proceedings of SPIN*, pages 113–130, 2000.
4 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. In *Proceedings of POPL*, pages 203–214, 2012.
5 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10, 2013.
6 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR*, pages 135–150, 1997.
7 Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proceedings of CONCUR*, pages 473–487, 2005.
8 Tomás Brázdil, Antonín Kucera, and Oldrich Strazovský. On the decidability of temporal properties of probabilistic pushdown automata. In *Proceedings of STACS*, pages 145–157, 2005.
9 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
10 Xiaojuan Cai and Mizuhito Ogawa. Well-structured pushdown systems. In *Proceedings of CONCUR*, pages 121–136, 2013.
11 Xiaojuan Cai and Mizuhito Ogawa. Well-structured pushdown system: Case of dense timed pushdown automata. In *Proceedings of FLOPS*, pages 336–352, 2014.
12 Christopher Earl, Ilya Sergey, Matthew Might, and David Van Horn. Introspective pushdown analysis of higher-order programs. In *Proceedings of ICFP*, pages 177–188, 2012.
13 Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of CAV*, pages 232–247, 2000.

14 Javier Esparza, Antonín Kucera, and Richard Mayr. Model checking probabilistic pushdown automata. In *Proceedings of LICS*, pages 12–21, 2004.

15 Javier Esparza, Antonín Kucera, and Stefan Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proceedings of TACS*, pages 316–339, 2001.

16 Javier Esparza and Stefan Schwoon. A bdd-based model checker for recursive programs. In *Proceedings of CAV*, pages 324–336, 2001.

17 Matthew Hague and C.-H. Luke Ong. Analysing mu-calculus properties of pushdown systems. In *Proceedings of SPIN*, pages 187–192, 2010.

18 Vineet Kahlon, Franjo Ivancic, and Aarti Gupta. Reasoning about threads communicating via locks. In *Proceedings of CAV*, pages 505–518, 2005.

19 Akash Lal, Thomas W. Reps, and Gogul Balakrishnan. Extended weighted pushdown systems. In *Proceedings of CAV*, pages 434–448, 2005.

20 Guoqiang Li, Xiaojuan Cai, Mizuhito Ogawa, and Shoji Yuen. Nested timed automata. In *Proceedings of FORMATS*, pages 168–182, 2013.

21 Xin Li and Mizuhito Ogawa. Conditional weighted pushdown systems and applications. In *Proceedings of PEPM*, pages 141–150, 2010.

22 Yasuhiko Minamide and Shunsuke Mori. Reachability analysis of the HTML5 parser specification and its application to compatibility testing. In *Proceedings of FM*, pages 293–307, 2012.

23 Thomas W. Reps, Stefan Schwoon, and Somesh Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *Proceedings of SAS*, pages 189–213, 2003.

24 Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *Proceedings of CAV*, pages 300–314, 2006.

25 Fu Song and Tayssir Touili. Efficient CTL model-checking for pushdown systems. In *Proceedings of CONCUR*, pages 434–449, 2011.

26 Fu Song and Tayssir Touili. PuMoC: a CTL model-checker for sequential programs. In *Proceedings of ASE*, pages 346–349, 2012.

27 Fu Song and Tayssir Touili. Model checking dynamic pushdown networks. In *Proceedings of APLAS*, pages 33–49, 2013.

28 Fu Song and Tayssir Touili. Efficient CTL model-checking for pushdown systems. *Theor. Comput. Sci.*, 549:127–145, 2014.

29 Fu Song and Tayssir Touili. Model checking dynamic pushdown networks. *Formal Asp. Comput.*, 27(2):397–421, 2015.

30 Yuya Uezato and Yasuhiko Minamide. Pushdown systems with stack manipulation. In *Proceedings of ATVA*, pages 412–426, 2013.