Reachability Analysis of Conditional Pushdown Systems with Patterns

Xin Li

1East China Normal University, Shanghai, 200062, China

Received XXX; accepted XXX; published online XXX

Abstract CPDSs (Conditional pushdown systems) extend pushdown systems by associating each pushdown transition rule with a regular language over the stack alphabet that specifies conditions under which the transition can be applied. Many program analysis and verification problems need to examine the runtime call stack of the program and can be reduced to the reachability problem of CPDSs. Unfortunately, the reachability problem of CPDSs is known to be EXPTIME-complete, given that the regular language associated with each pushdown transition is represented by a deterministic finite automaton. It indicates that there do not exist computationally tractable algorithms in general for tackling the problem. This work addresses the reachability problem for a subclass of CPDSs called patCPDS (CPDS with patterns), driven by the observation that the existing application instances of CPDSs carry regular languages in terms of regular expressions that obey certain patterns. We present new backward and forward saturation procedures for solving the reachability problems of patCPDSs, and further identify the subclass for which the reachability problem is solvable in fixed-parameter polynomial time, having applications in the verification problem for programs with stack inspection and without exception handling, and in the precise analysis and verification of object-oriented programs.

Keywords Verification, Reachability Problem, Conditional Pushdown Systems

1 Introduction

PDSs (pushdown systems) are well understood as abstract models of imperative programs with recursive procedures. By encoding the program as a PDS, procedure calls and returns are guaranteed to be correctly paired with one another in all runs of the system. This is the so-called context-sensitivity in terms of valid paths that one would expect for a precise program analysis and verification method. To check global security properties of programs with stack inspection, Esparza et al. introduce the model of PDSs with checkpoints [3] and it is reformulated as CPDSs (conditional pushdown systems) in [4]. In the extended model, CPDSs extend PDSs by associating each transition rule with a regular language over the stack alphabet, and the regular language specifies conditions under which the transition rule can be applied. Many program analysis and verification problems need investigate the program’s runtime call stack, and can be modelled and solved as the reachability problems of CPDSs [3-5].

Email: xinli@sei.ecnu.edu.cn
A classic and well-understood application of CPDSs is stack inspection. Access control policies are expressed in terms of permissions and are granted to codes grouped by different domains. Developers can set checkpoints in the code, and enforce runtime access control with the method CheckPermission(permission). At runtime, the method investigates the current call stack of the program for granted permissions until a privileged method is found. Let \( \Gamma \) be the stack alphabet. It is known that the stack inspection problem can be specified as regular expressions. The problem can be reduced to checking whether the current call stack satisfies the following regular expression: \( \text{TrustProc}^* (m_0 + m_1 + \cdots + m_n) \Gamma^* + \text{TrustProc}^* \), where we denote by \( \text{TrustProc} \) the set of trusted methods that hold the required permissions, and by \( \text{PrivProc} \) the set of privileged procedures with \( m_0, \ldots, m_n \in \text{PrivProc} \).

Another application of CPDSs is the reachability analysis of HTML5 parser specifications. Although a draft standard for HTML5 specifies detailed specifications of the parsing algorithm for HTML5 documents, there remain compatibility issues in the current implementations of web browsers because of the complexity of the specifications. In \cite{5}, Minamide and Mori formalize the HTML5 parser specification as an imperative programming language with commands for manipulating the entire stack, and reduce the problem to the reachability problem of CPDSs. In the model generated from the reachability analysis of HTML5 parser specification, pushdown transitions carry regular conditions such as \( \{ \text{Option}, \text{P}, \text{Rp}, \text{Ruby} \}^* \text{Li}^* + ! (\text{Dd} + \text{Option} + \text{Rp}) !^* \text{!Pt}^* \), where \( + \) denotes choices, \( ! \) denotes negation, \& denotes intersection, and \( \text{Option}, \text{P}, \text{Rp}, \text{Ruby} \), \( \text{Dd} \), \( \text{Li} \in \Gamma \).

PDSs are natural abstract models for procedure-oriented programs with recursive procedures. In \cite{4}, Li and Ogawa observe that OO (object-oriented) programs demand more elaborated context-sensitivity with respect to OO features for which PDSs are not directly applicable to model and tackle the analysis problem. For instance, to precisely model the program control flow with dynamic dispatch, one would have to look into the current call stack to check whether the calling context complies with that of the receiver object on which a method invocation is dispatch. When modeling OO languages with CPDSs, the stack can be used to encode the program calling contexts for instance in terms of call-site-strings. Consider finitely representing the calling contexts such that two calling contexts are identified iff their last \( k \) call sites are the same in order, similar to the abstraction of calling contexts used in \( k \)-CFA. Then pushdown transitions in this application scenario may carry regular conditions like \( m_1 m_2 \ldots m_k \Gamma^* \) where \( m_1, \ldots, m_k \) denote the last \( k \) call sites in order, for a given \( k \).

In \cite{3}, Esparza et al. prove that the reachability problem for PDSs with checkpoints, even for those with three control locations and no negative rules (that specifies the current stack content does not satisfy some regular language), is EXPTIME-complete. Derived from the result, the reachability problem of CPDSs is also EXPTIME-complete and computationally intractable. Thus one can only hope to seek efficient algorithms for those particular applications of CPDSs in practice that are not pathological. Yet, we notice that there exist polynomial formulations of context-sensitive \( k \)-CFA points-to analysis for OO languages \cite{2}, and efficient verification methods for programs with stack inspection \cite{6}. Then we observe that the aforementioned applications of CPDSs we are aware of carry regular languages in terms of regular expressions that obey certain patterns. A natural question is whether there exist tractable algorithms for solving the reachability problems of CPDSs for these practical applications.

In this work, we study the reachability problem of the subclass of CPDSs with regular languages over transition rules obeying certain patterns, and make the following contributions:

- The subclass of CPDSs, called patCPDS (CPDS with patterns), are proposed that are expressive enough to formulate existing applications of CPDSs found so far. The state and configuration reachability problems of patCPDSs are also introduced, respectively (Section 2).
- An extension of \( \mathcal{P} \)-automata, called \( \mathcal{S} \)-patterned \( \mathcal{P} \)-automata, is proposed to finitely represent regular sets of configurations of patCPDS (Section 4). In the new automaton, each state carries the set of signatures of any stack word that is accepted by the automaton from the state. And it can be judged readily whether a transition rule can be applied by comparing regular conditions with these signatures on each state, when saturating the automaton.
- The forward and backward saturation rules, as well as efficient implementation algorithms, are presented to solve the reachability problems of patCPDSs (Section 5). The proposed saturation rules
and implementation algorithms directly extend the classic ones for PDSs, and thus avoids an immediate exponential blowup of the system caused by making a detour to the reachability problems of PDSs.

- We further identify the subclass of patCPDSs for which the reachability problem is solvable in fixed-parameter polynomial time and space complexity. We hope this classification would be beneficial to readers who expects a tractable algorithm when formulating their problems as CPDSs.

The remainder of this paper is organized as follows: Section 2 recalls CPDSs, introduces the subclass of CPDSs with patterns, named patCPDS, and describes the problems to be addressed. Section 3 defines the signatures of stack words driven by patterns of regular languages, as well as operations and properties over signatures. Section 4 introduces the so-called Sig-patterned P-automata and an algorithm for constructing such an automaton given an ordinary P-automata. Section 5 presents forward and backward saturation procedures for solving the reachability problem of patCPDS, and presents an efficient implementation of the forward saturation procedure. Section 6 further identifies the subclass of patCPDS that is solvable with fixed-polynomial complexity for the reachability problem. Finally, Section 7 discusses related work and Section 8 concludes.

2 Preliminaries

2.1 Conditional Pushdown Systems

Conditional pushdown systems extend pushdown systems by further associating each pushdown transition with a regular language over the stack alphabet. The regular language specifies regular conditions under which the transition can be applied.

Definition 1. A CPDS (conditional pushdown system) $\mathcal{P}_c$ is a 4-tuple $(P, \Gamma, C, \Delta_c)$, where $P$ is a finite set of control locations, $\Gamma$ is a finite stack alphabet, $C$ is a finite set of regular languages over $\Gamma$, $\Delta_c \subseteq P \times \Gamma \times C \times P \times \Gamma^*$ is the set of transition rules. We write $\langle p, \gamma \rangle \xrightarrow{c} \langle q, \omega \rangle$ if $(p, \gamma, L, q, \omega) \in \Delta_c$. A configuration is a pair $\langle q, \omega \rangle$ with $q \in P$ and $\omega \in \Gamma^*$. A binary relation $\Rightarrow_c$ on configurations is defined such that $\langle p, \gamma \omega' \rangle \Rightarrow_c \langle q, \omega' \rangle$ for all $\omega' \in \Gamma^*$ if $(p, \gamma) \xrightarrow{c} \langle q, \omega \rangle$ and $\omega' \in L$. Given a set of configurations $S \subseteq P \times \Gamma^*$, we define $\text{cpre}^c(S) = \{ s' \in P \times \Gamma^* | \exists s \in S: s' \Rightarrow_c^* s \}$ and $\text{cpost}^c(S) = \{ s' \in P \times \Gamma^* | \exists s \in S: s \Rightarrow_c^* s' \}$ to be the (possibly infinite) set of predecessors and successors of $S$, respectively. The reachability problem of CPDSs in the simplest form asks, given a CPDS, whether a given configuration $\langle p', \omega' \rangle$ is reachable from a given configuration $\langle p, \omega \rangle$, i.e., does $\langle p, \omega \rangle \Rightarrow_c^* \langle p', \omega' \rangle$ hold?

Note that, if a transition rule is associated with $\Gamma^*$, it means that the rule can be used under any condition. Thus pushdown systems are special instances of CPDSs when each transition rule is associated with the regular condition $\Gamma^*$. Following the tradition of PDSs, we assume a normalized form of CPDSs where each transition $\langle p, \gamma \rangle \Rightarrow \langle q, \omega \rangle$ satisfies $|\omega| \leq 2$.

Theorem 1 (3). The reachability problem of CPDSs is EXPTIME-complete given that the regular language associated with each pushdown transition in CPDSs is represented by a DFA (deterministic finite automaton).

Theorem 1 indicates that there do not exist computationally tractable algorithms in general for tackling the reachability problem of CPDSs. Esparza et al. presented an algorithm that translates a CPDS to a corresponding PDS whereby they reduced the reachability problem of CPDSs to that of PDSs. Let $A_i$ be the DFA that recognizes the reversed language of $L_i \in C$ for each $i \in [1..m]$. They first build a product automaton $(\text{States}, A, \delta, s_0, F) \in \{ A_1, \ldots, A_m \}$, denoted by $\Pi_{i\in[1..m]}A_i$. Note that, $\Pi_{i\in[1..m]}A_i$ is also a DFA. The translation is given as follows, by synchronizing $\Pi_{i\in[1..m]}A_i$ and the underlying pushdown system of $\Pi_c$. For each transition rule on the left-hand side and for each state $r \in \text{States}$, we have

\[
\begin{align*}
\langle p, \gamma \rangle \xrightarrow{c} \langle q, \varepsilon \rangle & \quad \Rightarrow \quad \langle p, (\gamma, r) \rangle \Rightarrow \langle q, \varepsilon \rangle \\
\langle p, \gamma \rangle \xrightarrow{c} \langle q, \gamma' \rangle & \quad \Rightarrow \quad \langle p, (\gamma, r) \rangle \Rightarrow \langle q, (\gamma', r) \rangle \\
\langle p, \gamma \rangle \xrightarrow{c} \langle q, \gamma' \gamma'' \rangle & \quad \Rightarrow \quad \langle p, (\gamma, r) \rangle \Rightarrow \langle q, (\gamma', \text{t})(\gamma'', r) \rangle
\end{align*}
\]
where $\delta(r, \gamma') = t$. Then one can apply efficient reachability analysis of PDSs to solve the reachability problem of CPDSs. Note that, the size of the resulting pushdown system is $|\Delta_c| \times |\text{States}|$ since the product automaton is deterministic, and $|\text{States}|$ can be exponentially large.

At the heart of efficient model checking algorithms for PDSs are saturation algorithms over the so-called $\mathcal{P}$-automata. As given in Definition 2, a $\mathcal{P}$-automaton is a NFA (nondeterministic finite automaton) that finitely represents a possibly infinite set of configurations.

**Definition 2.** Given a CPDS $(P, \Gamma, C, \Delta_c)$, a $\mathcal{P}$-automaton $\mathcal{A} = (Q, \Gamma, \rightarrow, P, F)$ is a NFA, where $Q$ is the set of states, $\Gamma$ is the input alphabet, $\rightarrow \subseteq Q \times \Gamma \times Q$ is the set of transitions, and $P$ and $F$ are the set of initial and final states, respectively. We define $\rightarrow^* \subseteq Q \times \Gamma^* \times Q$ as the smallest relation satisfying the following rules: (i) $p \xrightarrow{\gamma} p$ for any $p \in Q$; (ii) $p \xrightarrow{\gamma \gamma'} p'$ if $(p, \gamma, p') \in \rightarrow$; (iii) $p \xrightarrow{\gamma \gamma'} p'$ if $p \xrightarrow{\gamma} p''$ and $p'' \xrightarrow{\gamma'} p'$ for some $p'' \in Q$. A configuration $p, \gamma$ is accepted by $\mathcal{A}$ if $p \xrightarrow{\gamma \gamma''} q$ for some $q \in F$. A set $C$ of configurations is accepted by $\mathcal{A}$ if each $c \in C$ is accepted by $\mathcal{A}$. A set $C$ of configurations is regular if it is accepted by some $\mathcal{P}$-automaton.

![Forward saturation rules for pushdown systems.](image)

We illustrate the forward saturation rules for PDSs in Figure 1. Our new algorithms for solving the reachability problem of CPDSs with patterns will be based on directly extending the $\mathcal{P}$-automata and its saturation rules.

### 2.2 Conditional Pushdown Systems with Patterns

**Definition 3.** A pattern, ranged over by $R$, is a regular expression defined in the following syntax,

$$\beta \text{ (basic pattern)} \in Ap := A^* | A^* \gamma_1 \ldots \gamma_k \Gamma^*$$

$$R \text{ (pattern)} \in R := \beta | ! R | R + R | R \& R$$

where $A \subseteq \Gamma$, $\gamma_1, \ldots, \gamma_k \in \Gamma$, and $k > 0$. The operator “!” denotes complementation, “&” denotes intersection, and “+” denotes union. The precedence of these operators is $\ast$ (Kleene-closure) $> \cdot$ (concatenation) $> ! > \& > +$. Here, $Ap$ denotes a set of basic patterns ranged over by $\beta$, and $R$ denotes a set of patterns ranged over by $R$. For any basic pattern $\beta \in Ap$, we write $\beta \in R$ if $\beta$ appears in $R$, and write $L(R)$ for the language of a pattern $R$.

By Definition 3, a pattern is a boolean combination of basic patterns, if we regard operators $!, \&$ and $+$ as logic negation, conjunction and disjunction, respectively, and interpret $Ap$ as atomic propositions. Without loss of generality, we always assume a pattern in DNF (disjunctive normal form) and in PNF (positive normal form) where negations appear only right before basic patterns.

**Definition 4.** A patCPDS (CPDS with patterns) $\mathcal{P}_c$ is a CPDS $(P, \Gamma, C, \Delta_c)$ where $C = \{R_1, \ldots, R_m\}$ is a finite set of patterns with $R_1, \ldots, R_m \in R$. A patterned configuration for some control location $p \in P$ and a pattern $R$, denoted by $(p, R)$, is the regular set $\{(p, \omega) | \omega \in R\}$ of configurations.

**Problems to be Addressed** Given a patCPDS $\mathcal{P}_c = (P, \Gamma, C, \Delta_c)$ where $C = \{R_1, \ldots, R_m\}$ and $R_i \in R$ for each $i \in [1..m]$, and given regular sets of configurations $S, T \subseteq P \times \Gamma^*$.

The following reachability problems of patCPDSs are to be addressed:

(i) **State Reachability**: given a control location $p \in P$, is $p$ forward (resp backwards) reachable from $S$, i.e., does there exist $\omega \in \Gamma^*$ such that $(p, \omega) \in cpost^*(S)$ (resp $(p, \omega) \in cpre^*(S)$)?
also observed in practice that the source and target configurations
consider a patCPDS
Example 1. The union of patterned configurations under this assumption.
Without loss of generality, we assume that satisfies this assumption, then we are able to simplify the last step in the reachability analysis algorithms of patterned configurations under this assumption.

Example 1. Consider a patCPDS $P_c = (P, \Gamma, C, \Delta_c)$, where $P = \{p_0, p_1, p_2\}$, $\Gamma = \{\gamma_1, \gamma_2\}$, $C = \{R_0, R_1, R_2\}$ with $R_0 = \Gamma^*$, $R_1 = \gamma_2^*\gamma_1^* + \gamma_2^*$ and $R_2 = \gamma_1^*\gamma_2^* + \gamma_2^*$, and $\Delta_c$ is given as follows:

$$
\Delta_c = \begin{cases}
\langle p_0, \gamma_1 \rangle \xrightarrow{R_0} \langle p_0, \gamma_2 \gamma_1 \rangle & \langle p_0, \gamma_2 \rangle \\
\langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \gamma_2 \gamma_2 \rangle & \langle p_0, \gamma_2 \rangle \\
\langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \gamma_2 \gamma_2 \rangle & \langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \epsilon \rangle \\
\langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \gamma_2 \gamma_2 \rangle & \langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \epsilon \rangle \\
\langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \gamma_2 \gamma_2 \rangle & \langle p_0, \gamma_2 \rangle \xrightarrow{R_0} \langle p_0, \epsilon \rangle \\
\end{cases}
$$

One may relate the above example to verification of programs with stack inspection (which only checks local security properties), where $\langle p_0, \gamma_2^* \rangle$ corresponds to the checkpoint inserted in the program, and $\langle p_1, \gamma_2^* \rangle$ and $\langle p_2, \gamma_2^* \rangle$ are the program points to which the program control will transfer when the stack inspection succeeds and fails, respectively.

Given the source configurations $S = \{p_0, R_3\}$ with $R_3 = \gamma_1^*$ and the target configurations $T = \{p_1, R_4\}$ with $R_4 = \gamma_2^*\gamma_1^*$. We are interested in whether $T$ is reachable from $S$. We will use this example as a running example in the rest of the paper.

3 Signatures of Stack Words

The analysis algorithm of CPDSs by making a detour to PDSs would result in an immediate blowup of transition rules in PDSs. To avoid it, we present saturation algorithms that directly extend the saturation rules of PDSs to patCPDSs. A key idea of our approach is to track signatures of the stack contents during saturation that only contain information relevant to the verification problem in question.

Driven by patterns, a signature of a stack word $\omega$ is pair $(\beta, \omega_{[1..k]})$ formed by

- a basic pattern $\beta$ involved in the verification problem that is satisfied by $\omega$, i.e., $\omega \in L(\beta)$, and
- its longest prefix within the length of some given $k \geq 0$, denoted by $\omega_{[1..k]}$.

Let $R = C \cup \{R_{m+1}, \ldots, R_k\} = \{R_1, \ldots, R_n\}$ be the set of (distinguished) patterns involved in the verification problem, and let $Ap(R)$ be the set of basic patterns that appear in $R$. Here, $k$ is the largest number in $\{n | A^* \gamma_1 \ldots \gamma_n \Gamma^* \in Ap(R)\}$. When $k = 0$, it indicates that the basic patterns in the form of $A^* \gamma_1 \ldots \gamma_k \Gamma^*$ are not involved in the verification problem, and $\omega_{[1..k]} = \epsilon$. We denote by $Sig = \mathbb{R} \times \bigcup_{0 \leq i \leq k} \Gamma^{(i)}$ the universe of signatures for the given patCPDS.

For an easy presentation, we denote a basic pattern by a pair $(A, a)$, and write $(A, \#)$ for $A^*$ and $(A, \gamma_1 \ldots \gamma_n)$ for $A^* \gamma_1 \ldots \gamma_n \Gamma^*$, respectively. Here, $\# \not\in \Gamma$ is a fresh symbol introduced to denote the bottom of the stack. In particular, $\Gamma^*$ is represented by $(\Gamma, \#)$ and denoted by $\beta_\Gamma$, and $\emptyset^* = \{\epsilon\}$ is represented by $(\emptyset, \#)$ and denoted by $\beta_\emptyset$. In the sequel, we do not distinguish a basic pattern and its pair representation when it is clear from the context. Next we define a partial order $\preceq \subseteq Ap \times Ap$ on basic patterns by, for any $(A, a), (A', a') \in Ap$,

$$(A, a) \preceq (A', a') \text{ if } A \subseteq A' \text{ and } a = a', \text{ or } (A', a') = \beta_\Gamma.$$
We will be only concerned with well-formed signatures in our algorithm. For any \( \omega \in \Gamma^* \), we prepare a function \([\cdot] : \Gamma^* \to 2^{\text{Sig}}\) that extracts the set of signatures from \( \omega \) by,

\[
[\omega] = \begin{cases} 
I_\varepsilon & \text{if } \omega = \varepsilon \\
\{(\beta, \omega_{[1..k]}) \mid \exists \beta \in \text{Ap}_R : \omega \in \mathcal{L}(\beta)\} & \text{otherwise}
\end{cases}
\]

where \( I_\varepsilon \overset{\text{def}}{=} \{(\varepsilon, \varepsilon)\} \). Obviously, \([\omega]\) is well-formed by definition.

**Example 2.** Let us revisit Example 1. We have that \( \text{Ap}_R = \{(\gamma_0, \#), \beta_1 : \{\gamma_2\}, \beta_2 : (\emptyset, \gamma_1), \beta_\tau\} \). It is not hard to see that \( \beta_2 \leq \beta_1 \), and that

\[
\begin{align*}
[\gamma_1] &= \{\beta_1, \beta_2, \beta_\tau\} = _\varepsilon \{\beta_2\}, \\
[\gamma_2] &= \{\beta_0, \beta_\tau\} = _\varepsilon \{\beta_0\}, \\
[\gamma_1 \gamma_2] &= \{\{\beta_1, \gamma_1\}, \{\beta_2, \gamma_1\}, \beta_\tau\} = _\varepsilon \{\{\beta_2, \gamma_1\}\}.
\end{align*}
\]

where set \(=_\varepsilon \) set \( =_\varepsilon \) means that set \( =_\varepsilon \) is an antichain of set with respect to \( _\varepsilon \).

Below we prepare two key functions upon which our new saturation algorithms for patCPDSs are built.

**Definition 5.** Let \( \text{match}(\beta) \overset{\text{def}}{=} \{\beta' \in \text{Ap}_R \mid \beta \leq \beta'\} \) for \( \beta \in \text{Ap}_R \). For any \( \gamma \in \Gamma \), and \( (\beta, \omega) \in \text{Sig} \) with \( \beta = (A, a) \), we define a **signature generation function** \( \text{gen}_{\text{Sig}}(\gamma, I) \) as follows:

\[
\text{gen}_{\text{Sig}}(\gamma, I) \overset{\text{def}}{=} \{\{\beta', (\gamma \omega)_{[1..k]}\} \mid \beta' \in \text{match}(A \cup \{\gamma\}, A) \cup \bigcup_{1 \leq j \leq k} \text{match}(\emptyset, (\gamma \omega)_{[1..j]})\}
\]

whereby we define a function \( \text{gen} : \Gamma \times 2^{\text{Sig}} \to 2^{\text{Sig}} \) such that, for any \( \gamma \in \Gamma \), and \( I \subseteq \text{Sig} \),

\[
\text{gen}(\gamma, I) \overset{\text{def}}{=} \bigcup_{(\beta, \omega) \in I} \text{gen}_{\text{Sig}}(\gamma, (\beta, \omega))
\]

By definition, one can easily conclude with the following fact that, given a well-formed set of signatures \( I \subseteq \text{Sig} \) and any symbol \( \gamma \in \Gamma \), \( \text{gen}(\gamma, I) \) will output a well-formed set of signatures.

**Lemma 1.** For any \( \gamma \in \Gamma \), \( I \subseteq \text{Sig} \), and \( \omega \in \Gamma^* \) such that \( I = [\omega] \), we have that \( \text{gen}(\gamma, I) = [\gamma \omega] \).

**Proof.** Consider any \( (\beta, \omega') \in \text{gen}(\gamma, I) \) with \( \beta = (A, a) \) and \( \omega' = (\gamma \omega)_{[1..k]} \). We show that \( \beta, \omega' \in [\gamma \omega] \). Let \( \text{Ap}_I \) be the set of basic patterns appearing in \( I \), i.e., \( \text{Ap}_I = \{\beta \mid (\beta, \omega) \in I\} \). By case analysis,

- if there exists \( (A', a') \in \text{Ap}_I \) such that \( (A, a) \in \text{match}(A' \cup \{\gamma\}, a') \), then it implies that \( (A' \cup \{\gamma\}, a') \leq (A, a) \), and thus \( A' \cup \{\gamma\} \subseteq A \) and \( a = a' \). Since \( I = [\omega] \), we know that \( \omega \in (A', a') \). Then \( \gamma \omega \in (A, a) \) and \( (\beta, \omega') \in [\gamma \omega] \).

- if \( (A, a) \in \text{match}(\emptyset, \omega'') \) with \( \omega'' = (\gamma \omega)_{[1..j]} \) for some \( 1 \leq j \leq k \), then \( (\emptyset, \omega'') \leq (A, a) \) and \( a = \omega'' \). Then \( \gamma \omega = \omega'' \omega_0 \in (A, a) \) for some \( \omega_0 \in \Gamma^* \), and \( (\beta, \omega') \in [\gamma \omega] \) trivially holds.

Consider any \( (\beta, \omega') \in [\gamma \omega] \) with \( \beta = (A, a) \) and \( \omega' = (\gamma \omega)_{[1..k]} \). We show that \( (\beta, \omega') \in \text{gen}(\gamma, I) \). By definition, we have \( \gamma \omega \in (A, a) \). Assume there exists \( \omega \in (A', a') \) such that \( a = a' \). Then \( A \supseteq A' \cup \{\gamma\} \) and \( (A' \cup \{\gamma\}, a') \leq (A, a) \). Thus \( (A, a) \in \text{match}(A' \cup \{\gamma\}, a') \) and \( (\beta, \omega') \in \text{gen}(\gamma, I) \). Otherwise, if there does not exist such an \( (A', a') \), then it means that \( a \) does not appear in \( \omega \). Thus we know that \( \gamma \omega = a \omega_0 \) for some \( \omega_0 \in \Gamma^* \) with \( 1 \leq |a| \leq k \). Then \( (A, a) \in \text{match}(\emptyset, (\gamma \omega)_{[1..j]}) \) for some \( 1 \leq j \leq k \) and \( (\beta, \omega') \in \text{gen}(\gamma, I) \).

**Example 3.** Consider Example 1. We have that

\[
[\gamma_1 \gamma_2] = \text{gen}(\gamma_1, \text{gen}(\gamma_2, I_\varepsilon)) = \text{gen}(\gamma_1, (\beta_0, \gamma_2)) = (\{\beta_1, \gamma_1\}, \{\beta_2, \gamma_1\}, \beta_\tau) = _\varepsilon \{(\beta_2, \gamma_1)\}
\]
Definition 6. For any pattern $R \in \mathcal{R}$, and any set of well-formed signatures $I \subseteq \text{Sig}$, we define an evaluation function $\text{eval} : \mathcal{R} \times 2^{\text{Sig}} \rightarrow \{\text{true}, \text{false}\}$ by,
\[
\text{eval}(R, I) \stackrel{\text{def}}{=} (\exists (S^+, S^-) \in \tau(R). S^+ \subseteq \text{Ap}(I) \wedge S^- \cap \text{Ap}(I) = \emptyset)
\]
where $\text{Ap}(I) = \{\beta' \mid \exists (\beta, \omega) \in I : \beta \subseteq \beta'\}$ returns the set of basic patterns appearing in $I$, and for any pattern $R$, $\tau(R)$ returns the least set satisfying the following rules:
\[
\begin{align*}
\tau(\beta) &\supseteq \{(\beta, \emptyset)\} \quad \tau(\emptyset) \supseteq \{\emptyset, \{\beta\}\} \\
\tau(R_1 \& R_2) &\supseteq (S^+_1 \cup S^-_1, S^+ \cup S^-_2) \mid \forall i \in \{1..2\}, (S^+_i, S^-_i) \in \tau(R_i) \\
\tau(R_1 + R_2) &\supseteq \tau(R_1) \cup \tau(R_2)
\end{align*}
\]
Lemma 2. For any pattern $R \in \mathcal{R}$ and any word $\omega \in \Gamma^*$, we have that $\omega \in \mathcal{L}(R)$ iff $\text{eval}(R, I) = \text{true}$ where $I = [\omega]$.
\begin{proof}
Suppose $\omega \in \mathcal{L}(R)$. By induction on the structure of $R$.
\begin{itemize}
\item Case $R = \beta \in A_p$: we have $\tau(R) = \{(\beta, \emptyset)\}$. Further, $\omega \in \mathcal{L}(R)$ implies that $\beta \in \text{Ap}(I[\omega])$. Then $\text{eval}(R, I) = \text{true}$ trivially holds.
\item Case $R = \emptyset$: we have $\tau(R) = \{\emptyset, \{\beta\}\}$. Further, $\omega \in \mathcal{L}(R)$ implies that $\beta \notin \text{Ap}(I[\omega])$. Then $\text{eval}(R, I) = \text{true}$ trivially holds.
\item Case $R = R_1 \& R_2$: by the definition of $\tau$ and $\text{eval}$, it is not hard to have that $\text{eval}(R_1 \& R_2, I) = \text{eval}(R_1, I) \wedge \text{eval}(R_2, I)$. Further, $\omega \in \mathcal{L}(R) \iff \omega \in \mathcal{L}(R_1)$ and $\omega \in \mathcal{L}(R_2)$. By induction hypothesis, $\text{eval}(R_1, I) = \text{true}$ and $\text{eval}(R_2, I) = \text{true}$. Then $\text{eval}(R, I) = \text{true}$.
\item Case $R = R_1 + R_2$: similarly, we have that $\text{eval}(R_1 + R_2, I) = \text{eval}(R_1, I) \lor \text{eval}(R_2, I)$ by definition. Further, $\omega \in \mathcal{L}(R) \iff \omega \in \mathcal{L}(R_1)$ or $\omega \in \mathcal{L}(R_2)$. Suppose that $\omega \in \mathcal{L}(R_1)$. By induction hypothesis, $\text{eval}(R_1, I) = \text{true}$. Then we have $\text{eval}(R, I) = \text{true}$.
\end{itemize}
Suppose that $\text{eval}(R, I) = \text{true}$ where $I = [\omega]$. By induction on the structure of $R$.
\begin{itemize}
\item Case $R = \beta \in A_p$: we have $\tau(R) = \{(\beta, \emptyset)\}$. Then $\text{eval}(R, I) = \text{true}$ implies that $\beta \in \text{Ap}(I)$. Thus $\omega \in \mathcal{L}(R)$.
\item Case $R = \emptyset$: we have $\tau(R) = \{\emptyset, \{\beta\}\}$. Then $\text{eval}(R, I) = \text{true}$ implies that $\beta \not\in \text{Ap}(I)$. Thus $\omega \notin \mathcal{L}(R)$.
\item Case $R = R_1 \& R_2$: we have $\text{eval}(R_1, I) = \text{true}$ and $\text{eval}(R_2, I) = \text{true}$. By induction hypothesis, $\omega \in \mathcal{L}(R_1)$ and $\omega \in \mathcal{L}(R_2)$, which implies that $\omega \in \mathcal{L}(R)$.
\item Case $R = R_1 + R_2$: we have $\text{eval}(R_1, I) = \text{true}$ or $\text{eval}(R_2, I) = \text{true}$. By induction hypothesis, $\omega \in \mathcal{L}(R_1)$ or $\omega \in \mathcal{L}(R_2)$, which implies that $\omega \in \mathcal{L}(R)$.
\end{itemize}
\end{proof}
Example 4. Let us revisit Example 1, and let $I = \{(\beta_1, \gamma_1)\}$. We have that $\tau(R_1) = \{(\beta_1), (\emptyset), (\beta_1, \emptyset)\}$, and $\tau(R_2) = \{(\emptyset, (\beta_0, \beta_1))\}$. Then we know that $\text{eval}(R_1, I) = \text{true}$ and $\text{eval}(R_2, I) = \text{false}$.

4 \textit{Sig-patterned }$\mathcal{P}$\textit{-automata}

The efficient saturation algorithms for solving the reachability problems of PDSs is based on the fact that, the set of predecessors and successors of any regular set of configurations is effectively closed, respectively, and $\mathcal{P}$-automata is used to finitely represent a (possibly infinite) regular set of configurations. We extend the approach to patCPDS and propose a so-called $\text{Sig}$-patterned $\mathcal{P}$-automata, ranged over by $\mathcal{B}$, in Definition 7, where each state $q = (q, i)$ in $\mathcal{B}$ carries a set of well-formed signatures $I \subseteq \text{Sig}$ such that $I = [\omega]$ for any $\omega \in \mathcal{L}(\mathcal{B}, q)$. Here, we write $\mathcal{L}(\mathcal{B}, q)$ for the set of words accepted by $\mathcal{B}$ from the state $q$ as usual. Without loss of generality, we require that $\mathcal{L}(A, q) \neq \emptyset$ for any non-final state in $\mathcal{B}$.

Definition 7. Let $\Gamma_{\epsilon} = \Gamma \cup \{\epsilon\}$ and let $\Omega$ be a finite set. A $\text{Sig}$-patterned $\mathcal{P}$-automaton $\mathcal{B} = (Q, \Gamma_{\epsilon}, \rightarrow, \text{Init}, F)$ for a given patCPDS $\mathcal{P}_c$ is a non-deterministic finite automaton, where $Q \subseteq \Omega \times 2^{\text{Sig}}$ is the finite set of states with $\Omega \supseteq \{q_{p, \gamma} \mid p \in P, \gamma \in \Gamma\}$, $\Gamma$ is the input alphabet, $\rightarrow \subseteq Q \times \Gamma_{\epsilon} \times Q$ is the finite set of transitions, $\text{Init} \subseteq (P \times 2^{\text{Sig}}) \cap Q$ is the set of initial states, and $F \subseteq (\Omega \times I_{\epsilon}) \cap Q$ is the set of final states. It satisfies that, for any $q = (s, i) \in Q$, $I = [\omega]$ for any word $\omega \in \mathcal{L}(\mathcal{B}, q)$. 
By convention, we write \( q \xrightarrow{\gamma} q' \) for \((q, \gamma, q') \in \rightarrow\), and denote by \( \rightarrow^* \) the reflexive and transitive closure of \( \rightarrow \). A configuration \((p, \omega)\) is accepted by \( B \) if \((p, I) \xrightarrow{\omega}^* f \) for some final state \( f \in F \) and \( I \subseteq \Sigma_1 \).

Here, \( \rightarrow^* \) is the least set of relations satisfying the following rules: (i) \( q \xrightarrow{\gamma}^* q \); (ii) \( q \xrightarrow{\gamma}^* q' \) if \( q \xrightarrow{\gamma} q' \); (iii) \( q \xrightarrow{\gamma}^* q' \) if \( q \xrightarrow{\gamma} q'' \) and \( q'' \xrightarrow{\gamma}^* q' \). We write \( \mathcal{L}(B) \) for the set of configurations accepted by \( B \).

Given a \( \Sigma_1 \)-patterned \( \mathcal{P} \)-automaton \( B \), we often denote by \( Q_B, \rightarrow_B, \text{Init}_B, F_B \) the states, transitions, initial states, and final states of \( B \), respectively.

By Definition 7, \( Q_B \) can be understood as the quotient sets of the states of the standard \( \mathcal{P} \)-automaton say \( A \), such that, for any state \( q \in A \), the quotient set of \( \mathcal{L}(A, q) \) by an equivalence relation \( \equiv \), denoted by \( \mathcal{L}(A, q)/\equiv \), is defined as \( \mathcal{L}(A, q)/\equiv \triangleq \{ \omega : \omega \in \mathcal{L}(A, q) \} \). Here, \( \omega \equiv \omega' \) if and only if \([\omega] = [\omega']\).

**Algorithm 1** Computing a \( \Sigma_1 \)-patterned \( \mathcal{P} \)-automaton \( B \) with \( \mathcal{L}(B) = S \)

**Require:** a \( \mathcal{P} \)-automaton \( A = (Q_A, \Gamma, \rightarrow_A, P, F_A) \) that recognises a regular set of configurations \( S \subseteq P \times \Gamma^* \), where \( A \) has no transitions into initial states \( P \) and no \( \varepsilon \)-transitions.

**Ensure:** a \( \Sigma_1 \)-patterned \( \mathcal{P} \)-automaton \( B = (Q_B, \Gamma, \rightarrow_B, \text{Init}_B, F_B) \) is output that recognises \( S \).

1. \( F_B := \{(f, I_0) | f \in F_A\} \);
2. \( \rightarrow_B := \emptyset \);
3. prepare a map \( l : Q_A \rightarrow 2^{Q_S} \);
4. for all \( q \in Q_A \setminus F_A \) do
5. \( l(q) := \emptyset \);
6. end for
7. for all \( q \in F_A \) do
8. \( l(q) := F_S \);
9. end for
10. \( \text{workset} := F_A; \text{done} := \emptyset \);
11. while \( \text{workset} \neq \emptyset \) do
12. take and remove a state \( q \) from \( \text{workset} \);
13. for all \( p \xrightarrow{\gamma} A q \) do
14. for all \( q' \in l(q) \) with \( q' = (q, I) \) do
15. if \( (p, \gamma, q') \notin \text{done} \) then
16. \( \text{done} := \text{done} \cup \{(p, \gamma, q')\} \);
17. \( l'(p) := \text{gen}(\gamma, I) \);
18. \( p' := (p, l') \);
19. \( \rightarrow_B := \rightarrow_B \cup \{p', \gamma, q'\} \);
20. if \( p' \notin \text{l}(p) \) then
21. \( \text{l}(p) := \text{l}(p) \cup \{p'\} \);
22. \( \text{workset} := \text{workset} \cup \{p\} \);
23. end if
24. end if
25. end for
26. end while
27. end for
28. \( Q_B := \bigcup_{(q, q') \in \rightarrow_B} \{q, q'\} \);
29. \( \text{Init}_B := Q_B \cap (P \times 2^{\Sigma_1}) \);
30. return \((Q_B, \Gamma, \rightarrow_B, \text{Init}_B, F_B)\);

Let \( A \) be an ordinary \( \mathcal{P} \)-automaton that recognizes the regular set \( S \) of configurations. Algorithm 1 takes as input \( A \), and outputs a \( \Sigma_1 \)-patterned \( \mathcal{P} \)-automaton \( B \) that recognizes \( S \). The main idea is to construct \( B \) that is backward bisimilar with \( A \) (the notion will be formally described below). To this end, we prepare a mapping \( l \) from the states in \( A \) to the power set of states in \( B \) (declared at Line 3) which induces a backward bisimulation relation between the two automata. Initially, \( l \) relates the final states \( F_A \) of \( A \) to the final states \( F_B \) of \( B \) (Line 7-9). Starting with the final states \( F_A \) as the workset, the algorithm takes a state \( q \) from the workset and traverses \( A \) backwards following the reverse of each transition \((p, \gamma, q) \in A \) (Line 13), and constructs a corresponding transition \((p', \gamma, q') \) in \( B \) with \( p' = (p, l') \) and \( l' = \text{gen}(\gamma, I) \), for each state \( q' = (q, I) \) in \( l(q) \) (Line 14-19). The state \( p \) will be added to the workset if \( l \) relates \( p \) to some newly-generated node \( p' \) in \( B \), and the while loop proceeds until no more states are to be processed in the workset. Note that, the algorithm maintains in \( \text{done} \) a set of triplets, e.g., \((p, \gamma, q')\) for the above, that have been already processed for making new transitions in \( B \), so as to ensure termination of the procedure.

To show the correctness of the above algorithm, we recall some established facts of simulation relations on finite automata. Let \( B_i = (Q_i, \Gamma_{\sim_i}, \rightarrow_{\sim_i}, \text{Init}_i, F_i) \) be a \( \Sigma_1 \)-patterned \( \mathcal{P} \)-automaton, for \( i \in \{0, 1\} \). A simulation relation between \( B_0 \) and \( B_1 \) is a relation \( \sim \subseteq Q_0 \times Q_1 \) such that \( q_0 \sim q_1 \) if (i) \( q_0 \in F_0 \) implies that \( q_1 \in F_1 \), and (ii) for any \( q_0 \xrightarrow{\gamma} q_0 p_0 \), there exists \( q_1 \xrightarrow{\gamma} p_1 \) such that \( p_0 \prec p_1 \). It is well-established that, given a simulation relation \( \sim \) on \( B_0 \) and \( B_1 \), \( q_0 \sim q_1 \) implies that \( \mathcal{L}(B_0, q_0) \subseteq \mathcal{L}(B_1, q_1) \). We say \( B_0 \) is simulated by \( B_1 \) if there exists a simulation relation \( \sim \subseteq Q_0 \times Q_1 \), such that, for any \( q_0 \in \text{Init}_0 \), there
exists \( q_1 \in \text{Init}_L \) such that \( q_0 \prec q_1 \). It follows that \( \mathcal{L}(B_0) \subseteq \mathcal{L}(B_1) \) if \( B_0 \) is simulated by \( B_1 \). Furthermore, we say \( B_0 \) and \( B_1 \) are bisimilar, denoted by \( B_0 \sim B_1 \), if \( B_0 \) is simulated by \( B_1 \), and vice versa. We have that \( \mathcal{L}(B_0) = \mathcal{L}(B_1) \) if \( B_0 \sim B_1 \).

Given a \( \text{Sig} \)-patterned \( \mathcal{P} \)-automaton \( B = (Q, \Gamma, \rightarrow, \text{Init}, F) \), we define a new automaton \( \hat{B} = (Q, \Gamma, \leftarrow, F, \text{Init}) \) by reversing \( \rightarrow \), i.e., \( \leftarrow = \{(q, \gamma, p) \mid (p, \gamma, q) \in \rightarrow\} \), and by exchanging the initial and final states of \( B \). Let \( \omega^{-1} \) be the reverse of a word \( \omega \in \Gamma^* \). It is not hard to see that \( \omega \in \mathcal{L}(B) \) if and only if \( \omega^{-1} \in \mathcal{L}(\hat{B}) \).

**Lemma 3.** Algorithm 1 has the following properties:

i) For \( q \in Q_A \), and \( q' \in Q_B \), we write \( q \prec_{AB} q' \) if \( q' \in l(q) \). Then \( \prec_{AB} \) is a simulation relation between \( \hat{A} \) and \( \hat{B} \), and \( \hat{A} \) is simulated by \( \hat{B} \).

ii) For \( q \in Q_A \), and \( q' \in Q_B \), we write \( q' \sim_{BA} q \) if \( q' \in l(q) \). Then \( \sim_{BA} \) is a simulation relation between \( \hat{B} \) and \( \hat{A} \), and \( \hat{B} \) is simulated by \( \hat{A} \).

iii) \( A \) and \( B \) are bisimilar, i.e., \( A \sim B \).

**Proof.** We show that \( \hat{A} \) is simulated by \( \hat{B} \). It immediately follows the algorithm construction at Line 13-20 that, for any \( p \xrightarrow{} A q \), for any \( q' \in l(q) \), there exists \( p' \xrightarrow{} B q' \) such that \( p' \in l(p) \) and \( p' = (p, I) \) for some \( I \). Line 5 in the algorithm relates initial states of \( A \) and \( B \). We have that \( \prec_{AB} \) is a simulation relation between \( A \) and \( B \). One can similarly conclude with ii) that \( B \) is simulated by \( A \) by the algorithm construction. iii) follows immediately from i) and ii).

**Theorem 2.** If Algorithm 1 takes as input a \( \mathcal{P} \)-automaton \( A \) that recognizes a regular set \( S \) of configurations where \( A \) has no transitions into the initial states and no \( \varepsilon \)-transitions, then it outputs a \( \text{Sig} \)-patterned \( \mathcal{P} \)-automaton \( B \) that recognizes \( S \), and \( B \) has no transitions into the initial states and no \( \varepsilon \)-transitions.

**Proof.** It immediately follows from Lemma 3 that \( \mathcal{L}(B) = S \), and that \( B \) has no transitions into the initial states and no \( \varepsilon \)-transitions, given that \( A \) satisfies those conditions and \( A \sim B \). In addition, the number of the states of \( B \) is bounded by \( |(P \cup Q_A) \times 2^{\text{Sig}}| \), and lines 15 guarantees that the same processing will not be repeated. Then the algorithm terminates.

We show that \( B \) is a \( \text{Sig} \)-patterned \( \mathcal{P} \)-automaton during the algorithm. By induction on the number \( i \) of transitions added to \( B \).

- Case \( i = 0 \): there are only final states in \( B \) and the property trivially holds.
- Case \( i > 0 \): suppose that the algorithm adds a new transition \( (p', \gamma, q') \) to \( B \) at line 19. We denote by \( B' \) the updated automaton after line 19. By induction hypothesis, \( I = [\omega] \) for any \( \omega \in \mathcal{L}(B, q') \). Let \( \text{newl} = \{\gamma \omega \mid \omega \in \mathcal{L}(B, q')\} \). Then \( \mathcal{L}(B', p') = \text{newl} \cup \mathcal{L}(B, p') \). By Lemma 1, we have that \( I' = [\gamma \omega] \). If \( p' \) is a new state, then \( \mathcal{L}(B', p') = \text{newl} \) and the property follows immediately. Otherwise, it immediately follows from the induction hypothesis that, \( I' = [\omega'] \) for any \( \omega' \in \mathcal{L}(B, p') \).

**Theorem 3.** Algorithm 1 takes \( O(|\rightarrow_A| \cdot n_0^2) \) time and space where \( n_0 = 2^{|Ap|} \cdot |\Gamma|^k \).

**Proof.** By choosing appropriate data structures, all the needed membership test, removal operations, etc., can take constant time. By a simple analysis, we know that each triplet \( (p, \gamma, q') \in \text{done} \) is processed only once in the algorithm. Thus the number of Line 19 is executed in \( O(|\rightarrow_A| \cdot n_0^2) \) times with \( n_0 = 2^{|	ext{Sig}|} \) and \( |	ext{Sig}| = |Ap| \cdot |\Gamma|^k \). Since only well-formed signatures are considered, \( n_0 \) can be reduced to \( 2^{|Ap|} \cdot |\Gamma|^k \). Thus we conclude the complexity results as stated in the theorem.

**Example 5.** Consider Example 1 again. The \( \mathcal{P} \)-automaton \( A \) that accepts \( S = \{ (p_0, \gamma_1 \Gamma^*) \} \) is given in Figure 2, and the \( \text{Sig} \)-patterned \( \mathcal{P} \)-automaton \( B \) that accepts \( S \) is given in Figure 3. computed by using Algorithm 1 given \( A \). Here the nodes of double circles denote final states, and initial states are labelled with starting arrows.
5 Saturation Algorithms for Reachability Analysis of patCPDSs

5.1 Forward and Backward Saturation Rules

Given a Sig-patterned P-automaton B recognizing a regular set of configurations C, the forward saturation rules for computing a Sig-patterned P-automaton that recognize cpost(C) are given as follows:

- If \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \varepsilon \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\gamma}^* (q, I_2) \) in the current automaton, and \( \text{eval}(R, I_2) = \text{true} \), add transition \( (p', I_2, \varepsilon, (q, I_2)) \);
- If \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \gamma' \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\gamma'}^* (q, I_2) \) in the current automaton, and \( \text{eval}(R, I_2) = \text{true} \), add transitions \( (p', I_1, \gamma', (q, I_2)) \) where \( I = \text{gen}(\gamma', I_2) \);
- If \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \gamma'' \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\gamma''}^* (q, I_2) \) in the current automaton, and \( \text{eval}(R, I_2) = \text{true} \), add transitions \( (p', I', \gamma', (q, \gamma', I)) \) and \( (q, \gamma', I_2) \) where \( I_2 = \text{gen}(\gamma'', I_2) \) and \( I' = \text{gen}(\gamma', I) \).

Intuitively, suppose there exist \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \omega \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\gamma}^* (q, I_2) \) in the current automaton \( B_i \). Consider any \( \omega' \in L(B_i, (q, I_2)) \). We know that \( \langle p, \omega \omega' \rangle \in \text{cpost}^*(C) \). If \( \text{eval}(R, I_2) = \text{true} \), it follows from Lemma 2 that \( \omega' \in L(R) \). Thus, the rule above can be applied here. Then new transitions are added to the automaton \( B \) so as to ensure \( \langle p', \omega \omega' \rangle \in \text{cpost}^*(C) \).

Similarly, given a Sig-patterned P-automaton B recognizing a regular set of configurations C, the backward saturation rules for computing a Sig-patterned P-automaton that recognizes cpre*(C) are given as follows:

- If \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \omega \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\omega}^* (q, I_2) \) in the current automaton, and \( \text{eval}(R, I_2) = \text{true} \), add transitions \( (p, I_1, \gamma, (q, I_2)) \) where \( I = \text{gen}(\gamma, I_2) \).

Intuitively, suppose there exist \( \langle p, \gamma \rangle \xrightarrow{R} \langle p', \omega \rangle \in \Delta_c \) and \( (p, I_1) \xrightarrow{\omega}^* (q, I_2) \) in the current automaton \( B \). Consider any \( \omega' \in L(B, (q, I_2)) \). We know that \( \langle p, \gamma \omega' \rangle \in \text{cpre}^*(C) \). If \( \text{eval}(R, I_2) = \text{true} \), it follows from Lemma 2 that \( \omega' \in L(R) \). Thus, the rule above can be applied here. Then new transitions are added to the automaton \( B \) so as to ensure \( \langle p, \gamma \omega' \rangle \in \text{cpre}^*(C) \).

5.2 An Efficient Algorithm of Forward Saturation

In Algorithm 2, we present an efficient implementation of the forward saturation procedure, by directly extending the efficient forward saturation algorithm of pushdown systems (cf. Algorithm 2 in Figure 3.4, Section 3.7). The algorithm starts with processing transitions in \( B \) leading from the initial states and applies the saturation rules iteratively. Lines 23-24 and Lines 26-27 are added to process \( \varepsilon \)-transitions. The extension here is generating signatures for making the source states of the new transition added to
the automaton at Lines 13, 18, 19, and safely judging whether a rule can be applied at Lines 9, 12 and 17. The correctness of Algorithm 2 is given in Theorem 4. An implementation algorithm of the backward saturation procedure can be similarly constructed (cf. Algorithm 1 in Figure 3.3, Section 3 [7]).

Algorithm 2 Computing a $\text{Sig}$-patterned $\mathcal{P}$-automaton $B_t$ with $\mathcal{L}(B_t) = cpost^*(S)$

Require: a $\text{Sig}$-patterned $\mathcal{P}$-automaton $B = (Q_B, \Gamma, \rightarrow_B, \text{Init}_B, F_B)$ that recognises $S$, where $B$ has no transitions into the initial states and no $\varepsilon$-transitions.

Ensure: a $\text{Sig}$-patterned $\mathcal{P}$-automaton $B_t = (Q_{B_t}, \Gamma_{\varepsilon}, \rightarrow_{B_t}, \text{Init}_{B_t}, F_{B_t})$ that recognises $cpost^*(S)$.

1: $ws := \rightarrow_B \cap (\text{Init}_B \times \Gamma \times Q_B)$
2: $\rightarrow_{B_t} := \rightarrow_B \setminus ws$
3: while $ws \neq \emptyset$ do
4: take and remove a transition $t$ from $ws$ with $t = ((p, I_1), \gamma, (q_0, I_2))$
5: if $t \notin \rightarrow_{B_t}$ then
6: $\rightarrow_{B_t} := \rightarrow_{B_t} \cup \{t\}$
7: if $\gamma \neq \varepsilon$ then
8: for all $p, \gamma \xrightarrow{R} (p', \varepsilon) \in \Delta_c$ do
9: if $\text{eval}(R, I_2) = \text{true}$ then
10: $ws := ws \cup \{(p', I_2, \varepsilon, (q_0, I_2))\}$
11: end if
12: end for
13: for all $p, \gamma \xrightarrow{R} (p', \gamma') \in \Delta_c$ do
14: if $\text{eval}(R, I_2) = \text{true}$ then
15: $I = \text{gen}(\gamma', I_2)$
16: $ws := ws \cup \{(p', I, \gamma', (q_0, I_2))\}$
17: end if
18: end for
19: end if
else$
20$: if $\text{eval}(R, I_2) = \text{true}$ then
21: $I = \text{gen}(\gamma''_0, I_2)$
22: $I' = \text{gen}(\gamma, I)$
23: $ws := ws \cup \{((p', I), \gamma', (q_0, I_2))\}$
24: $\rightarrow_{B_t} := \rightarrow_{B_t} \cup \{((q_0, \gamma), \gamma', (q_0, I_2))\}$
25: for all $((p', I), \varepsilon, (q_0', \gamma', I))$ do
26: $ws := ws \cup \{((p', I), \gamma'', (q_0, I_2))\}$
27: end for
28: end if
29: end for
30: else
31: for all $((q_0, \gamma), (q_0', \gamma')) \in \rightarrow_{B_t}$ do
32: $ws := ws \cup \{((p, I_1), \gamma', (p_0', I_3))\}$
33: end for
34: end if
35: end if
36: end while
37: $Q_{B_t} := \bigcup_{(q, \gamma, q') \in \rightarrow_{B_t}} \{q, q'\}$
38: $\text{Init}_{B_t} := Q_{B_t} \cap (P \times 2^{S_{\text{Sig}}})$
39: return $(Q_{B_t}, \Gamma_{\varepsilon}, \rightarrow_{B_t}, \text{Init}_{B_t}, F_{B_t})$

Theorem 4. If Algorithm 2 takes as input a $\text{Sig}$-patterned $\mathcal{P}$-automaton $B$ that recognises $S$, where $B$ has no transitions into the initial states and no $\varepsilon$-transitions, then it outputs a $\text{Sig}$-patterned $\mathcal{P}$-automaton $B_t$ that recognises $cpost^*(S)$.

Proof. Since $Q_{B_t} \subseteq \Omega \times 2^{S_{\text{Sig}}}$ is finite, only finitely many transitions can be added to $B$ and the algorithm terminates by the condition at Line 5.

First, we show that $B_t$ is always a $\text{Sig}$-patterned $\mathcal{P}$-automaton during the algorithm. That is, for any $q = (s, I) \in Q_{B_t}$, $I = [\omega]$ for any $\omega \in \mathcal{L}(B_t, q)$. By induction on the number $i$ of transitions added to $B_t$.

- Case $i = 0$: $B_t = B \setminus ws$ and the property trivially holds given that $B$ is a $\text{Sig}$-patterned $\mathcal{P}$-automaton.
- Case $i > 0$: suppose that the algorithm adds a new transition $t$ to $B_t$. This can occur at Line 6 or 24, and the former is due to the update of workset $ws$ at Lines 10, 16, 23, 26, and 32, respectively. For the cases at Lines 10, 26, and 32, the proof is trivial, because these cases tackle with the situation when $\varepsilon$-transitions are involved and do not introduce any new signatures and words into $B_t$. Let $B_t'$ be the updated automaton after adding the transition. The proof for the cases at Line 16 and 23 are similar by using Lemma 1. Consider Line 16. For any $\omega \in \mathcal{L}(B_t, (q_0, I_2))$, we have $I_2 = [\omega]$ by induction hypothesis. By Lemma 1, we have $I = [\gamma'\omega]$. Let $\text{newl} = \{\gamma'\omega \mid \omega \in \mathcal{L}(B_t, (q_0, I_2))\}$. Then $\mathcal{L}(B_t', (p', I)) = \text{newl} \cup \mathcal{L}(B_t, (p', I))$. If $(p', I)$ is a newly added state, then $\mathcal{L}(B_t', (p', I)) = \text{newl}$ and the property follows immediately. Otherwise, it follows from the induction hypothesis that, $I = [\omega']$ for any $\omega' \in \mathcal{L}(B_t', (p', I))$.

To show $\mathcal{L}(B_t) = cpost^*(S)$, we show that $\mathcal{L}(B_t) \supseteq cpost^*(S)$ and $\mathcal{L}(B_t) \subseteq cpost^*(S)$, respectively.
Since $c\text{post}^*(S) = \bigcup_i c\text{post}^i(S)$, to show $\mathcal{L}(B_i) \supseteq c\text{post}^*(S)$ amounts to showing that $\mathcal{L}(B_i) \supseteq c\text{post}^i(S)$ for any $i \geq 0$. By induction on $i$.

- Case $i = 0$: the proof is trivial since $B_i \supseteq \mathcal{B}$ and $\mathcal{B}$ accepts $S$.

- Case $i > 0$: Suppose $\mathcal{L}(B_i) \supseteq c\text{post}^i(S)$ by induction hypothesis. Consider any $((p, I_1), \gamma, (q_0, I_2)) \in \rightarrow_{\mathcal{B}_i}$ such that $(p, \gamma \omega) \in \mathcal{L}(B_i) \cap c\text{post}^i(S)$ for some $\omega \in \mathcal{L}(B_i, (q_0, I_2))$ and $\gamma \neq \varepsilon$.

If there is a pop transition $(p, \gamma) \xrightarrow{R} (p', \varepsilon) \in \Delta$ and $\omega \in R$, then the rule can be applied here and $(p', \omega) \in c\text{post}^{i+1}(S)$. By Lemma 2, we have $\text{eval}(R, I_2) = \text{true}$. By the algorithm construction as shown at Line 8-10, we have $(p', \omega) \in \mathcal{L}(B_i)$, obviously. We can apply the similar reasoning above to other kinds of transition rules, corresponding to Line 13 and 19 in the algorithm, and conclude that $\mathcal{L}(B_i) \supseteq c\text{post}^{i+1}(S)$.

To show $\mathcal{L}(B_i) \subseteq c\text{post}^*(S)$ amounts to showing that $\omega \in \mathcal{L}(B_i)$ implies $\omega \in c\text{post}^*(S)$ for any $\omega \in \Gamma^*$. We proceed by induction on the number $i$ of transitions added to $B_i$.

- Case $i = 0$: the proof is trivial since $B_i \subseteq \mathcal{B}$ and $\mathcal{B}$ accepts $S$.

- Case $i > 0$: Suppose that the algorithm adds a new transition $t$ to $B_i$ at Line 10 resulting in $(p', \omega) \in \mathcal{L}(B_i)$. It also implies that $\text{eval}(R, I_2) = \text{true}$. By Lemma 2, we have that $\omega \in R$ for some $\omega \in \mathcal{L}(B_i, (q_0, I_2))$. By induction hypothesis, we know that $(p, \gamma \omega) \in c\text{post}^*(S)$. Thus the rule at Line 8 can be applied to $(p, \gamma \omega)$ resulting in $(p', \omega) \in c\text{post}^*(S)$. Similarly, we can prove the other cases when a transition is added at Lines 16 and 23-24. Note that, Line 26 and Line 32 do not matter, for they tackle with $\varepsilon$-transitions and do not introduce any new word by $\mathcal{L}(B_i)$. The theorem is finally proved. $\Box$

**Theorem 5.** Let $n_0$ be the number of different pairs $(p, \gamma)$ such that there exists $(p, \gamma) \rightarrow (p', \gamma' \gamma'')$ in $\Delta$. Algorithm 2 takes $O((|P| \cdot |\Delta_c| \cdot (|Q_{\mathcal{B}}| + P) + n_0) + |P| \cdot |\rightarrow_\mathcal{B}| \cdot n_1)$ time and space, where $n_1 = 2(|\mathcal{B}| \cdot |\Gamma|^k)$.

**Proof.** Algorithm 2 extends the efficient forward saturation algorithm of pushdown systems (cf. Algorithm 2 of Section 3 in Figure 3.4 and Lemma 3.11 in [7]). The space and time complexity of the algorithm is increased by a factor of $2^{|\text{Sig}|}$ where $|\text{Sig}| = |\mathcal{A}_p| \cdot |\Gamma|^k$. Given the fact that only well-formed signatures are concerned in the analysis, we can reduce $|\text{Sig}|$ to be $2^{|\mathcal{A}_p|} \cdot |\Gamma|^k$. $\Box$

### 5.3 Overall Reachability Analysis Algorithm of patCPDS

The pattern-driven forward (resp. backward) saturation algorithm of patCPDS consists of the following major steps:

1. First, as given in Algorithm 1, it takes as input the $\mathcal{P}$-automaton $\mathcal{A}$ that recognises the regular set $S$ of source configurations (resp. $T$ of target configurations), and outputs a $\text{Sig}$-patterned $\mathcal{P}$-automaton $\mathcal{B}$ that recognises $S$ (resp. $T$).

2. Next, it takes as input the $\text{Sig}$-patterned $\mathcal{P}$-automaton $\mathcal{B}$ computed in the first step, and iteratively applies the forward (resp. backward) saturation rules on $\mathcal{B}$ until convergence. Upon termination, it outputs a $\text{Sig}$-patterned $\mathcal{P}$-automaton $\mathcal{B}_f$ (resp. $\mathcal{B}_b$) that recognises $c\text{post}^*(S)$ (resp. $c\text{pre}^*(T)$).

3. Finally, we are ready to determine,

- State reachability: for any $p \in P$, $p$ is forward reachable from $S$ (resp. backward reachable from $T$) iff there exists $(p, I) \in \text{Init}_{\mathcal{B}_i}$ (resp. $(p, I) \in \text{Init}_{\mathcal{B}_f}$) for some $I \subseteq \text{Sig}$.

- Configuration reachability:

  (i) If $T$ is a finite union of patterned configurations, then $T$ is reachable from $S$ iff there exists $(p, I) \in \text{Init}_{\mathcal{B}_i}$ and $(p, R) \in T$ (resp. $(p, I) \in \text{Init}_{\mathcal{B}_f}$ and $(p, R) \in S$) such that $\text{eval}(R, I) = \text{true}$, and $T$ is unreachable from $S$, otherwise.

  (ii) Else, $T$ is reachable from $S$ iff $\mathcal{B}_i \cap \mathcal{A}_T \neq \emptyset$ (resp. $\mathcal{B}_b \cap \mathcal{A}_S \neq \emptyset$) and unreachable otherwise.

The correctness of (i) in the last step is shown in Theorem 6. It is straightforward to conclude the correctness for other cases.
Theorem 6. Let $B = (Q, \Gamma, \rightarrow, \text{Init}, F)$ be a $\text{Sig}$-patterned $\mathcal{P}$-automaton such that $\mathcal{L}(B) = \text{cpost}^*(S)$ (resp $\mathcal{L}(B) = \text{cpre}^*(T)$). Then we have that, $T$ is reachable from $S$ iff there exists some $(p, I) \in \text{Init}$ and $(p, R) \in T$ (resp $(p, R) \in S$) such that $\text{eval}(R, I) = \text{true}$.

Proof. We consider the forward saturation algorithm and prove the two directions as follows. The backward saturation algorithm can be similarly proved.

- Suppose there exists $(p, I) \in \text{Init}$ and $(p, R) \in T$ such that $\text{eval}(R, I) = \text{true}$. Since $\mathcal{L}(B) = \text{cpost}^*(S)$, for any $(p, I) \in \text{Init}$, there exists $(p, \omega) \in \text{cpost}^*(S)$ such that $I = [\omega]$. Since $\text{eval}(R, I) = \text{true}$, we have that $\omega \in R$ by Lemma 2. It follows that $(p, \omega) \in T$, and $T$ is reachable from $S$.

- Suppose that $T$ is reachable from $S$. Then by definition, $T \cap \text{cpost}^*(S) \neq \emptyset$ and there exists $(p, \omega) \in T \cap \text{cpost}^*(S)$. Since $T$ is a finite union of patterned configurations, there exists $(p, R) \in T$ such that $\omega \in R$. Since $\mathcal{L}(B) = \text{cpost}^*(S)$, there exists $(p, I) \in \text{Init}$ such that $I = [\omega]$. It follows from Lemma 2 that $\text{eval}(R, I) = \text{true}$. \hfill $\square$

Example 6. Consider Example 1. Figure 4 illustrates the $\text{Sig}$-patterned $\mathcal{P}$-automaton $B_f$ that recognizes $\text{cpost}^*(S)$ by saturating the automaton $B$ as given in Figure 3 where dashed edges are newly-added during the forward saturation.

### 6 Subclass with Fixed-Parameter Polynomial Complexity

We have presented new reachability analysis algorithms for patCPDS by directly extending the saturation algorithms of pushdown systems over $\mathcal{P}$-automata. However, the algorithms still exhibit an exponential runtime in the size of signatures $|\text{Sig}|$. In this section, we further identify the subclass for which the reachability problem is solvable in fixed-parameter polynomial time.

We further divide patterns into two categories. A pattern is simple if it does not contain & and ! operators but basic patterns and the + operator, and strict otherwise. Correspondingly, we say a patCPDS is simple if it only contains simply patterns, and strict otherwise. The aforementioned compatibility checking of HTML5 parser specifications is an instance of strict patCPDS. The other applications of patCPDSs that we are aware of, including the verification problem for programs with stack inspection and without exception handling, and precise call graph construction (equivalently, points-to analysis) for Java programs, are simple.

Recall the evaluation function in Definition 6. If a pattern $R$ is simple, then by definition, $\text{eval}(R, I)$ for some given signatures $I \subseteq \text{Sig}$ amounts to checking whether any basic pattern belongs to $I$ also appears in $R$. Then it will be enough for each state to track and check the signatures of the stack words separately in the $\text{Sig}$-patterned $\mathcal{P}$-automaton, i.e., each state in the automaton only have to carry a single signature it admits. The modification on the saturation algorithms is minimal: any transition $((p, I_1), \gamma, (q, I_2))$ to be added to the $\text{Sig}$-patterned $\mathcal{P}$-automaton will be split into a set of transitions $\{(p, ((\beta, \omega))), (q, ((\beta', \omega'))) \mid (\beta, \omega) \in I_1, (\beta', \omega') \in I_2\}$ before adding to the automaton. The correctness of the algorithms are preserved, trivially. Yet we obtain fixed-parameter polynomial complexity for the subclasses as summarized in Table 1, where it also compares the complexity with the
algorithm that takes a detour to reachability problems of PDSs (as briefly introduced in Section 2). Here, it only shows the factor in which the time and space complexity of the saturation algorithms of PDSs will be polynomially increased, for the algorithm taking a detour to PDSs in the second column and for the new saturation algorithm presented in this work in the third column, respectively.

<table>
<thead>
<tr>
<th>patCPDS</th>
<th>Detour to PDS</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td>strict</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simple with $k &gt; 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simple with $k = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The second row shows the subclass for which the reachability problem is solvable in fixed-parameter polynomial time (in $|\Gamma|^k$). Notably, the last row shows the simplest subclass of patCPDS where the patCPDS is simple and only contains basic patterns in the form of $A^*$ and the + operator. If one uses the algorithms that takes a detour to pushdown systems to answer the reachability problem of it, the time and space complexity remains exponential in the worst case (consider that the algorithm generates $|\text{States}|$ to blow up). In contrast, we would obtain a tractable algorithm by directly saturating $\text{Sig}$-patterned $\mathcal{P}$-automata.

7 Related Work

The model checking techniques of PDSs and a lot of variants are extensively studied, and we limit our focus to those closely related to CPDSs. An extensively-studied problem that originates CPDSs, is model checking on security properties of programs with stack inspection. The problem of interest is whether local security checks in the code are sufficient enough to ensure a global security property. In [1], Besson et al. applied LTL to specify a class of global security properties defined over the program’s control flow graph, and proposed a model checking technique for the class of properties by using finite automata to encode the stack contents. Their work motivates a few works including [3, 6].

In [6], Nitta et al. introduced a practically important subclass of programs with stack inspection, and showed that the class is solvable efficiently (sometime linear in the size of a program given that the regular language involved in the analysis is represented by DFA). Note that, the studied subclass of stack inspection does not include exception handling when access control fails.

Esparza et al. further proposed model checking techniques that can check general LTL properties on PDSs with regular valuations over the entire stack contents of configurations [3], and provided a detailed study of the complexity analysis of the problem. The authors also introduced the model called PDSs with checkpoints to formulate the verification problem for programs with stack inspection, which is the pioneering work of CPDSs. In particular, the authors proved that, the reachability problem for PDSs with checkpoints, even for those with three control locations and no negative rules, is EXPTIME-complete.

In [4], Li and Ogawa reformulated the model of PDSs with checkpoints as CPDSs, to more readily model context-sensitive program analysis and verification problems for Java, and presented precise call graph construction, equivalently, context-sensitive points-to analysis, as a new application of CPDSs. Note that, their problem is compactly formulated as weighted PDSs, yet can be trivially reformulated as reachability problem of CPDSs since only a finite data domain (i.e., a finite abstraction on heap objects) is involved in their application. The authors showed that a precise analysis and verification method for OO programs is beyond the capability of PDSs under a direct application.

The original algorithm for solving the reachability problem of CPDSs is an offline algorithm that first translates CPDSs to PDSs by synchronizing the underlying PDS and the product of DFAs accepting regular conditions [3]. The translation, however, can cause an exponential blow-up of the system.

Minamide and Mori studied reachability analysis of HTML5 parser specifications as a new application of CPDSs and proposed a new backward saturation algorithm for CPDSs. By extending $\mathcal{P}$-automata with
regular lookahead where each transition carries a regular expression under which the transition step can be made, their saturation algorithm computes relatives and intersection of regular expressions for computing the predecessor configurations. They do not give any complexity analysis yet provide extensive empirical results on reachability analysis of HTML5 parser specifications. Uezato and Minamide extended the realm of CPDSs by introducing PDSs with transductions (TrPDSs) in which each transition is associated with a transducer and the entire stack content can be modified. TrPDSs are Turing-complete, unsurprisingly. The authors further introduced finite TrPDSs for which the reachability problem is decidable, as well as saturation algorithms for solving the problem. Their work is recently improved by Song et al. in [8], where the authors proposed two new efficient saturation algorithms that directly extend the saturation algorithms over \( \mathcal{P} \)-automata.

In [9], Vy and Li presented an on-the-fly model checking algorithm for weighted CPDSs, by interleaving saturation procedures for computing successor configurations with regular condition checking, and by synchronizing the underlying PDS with DFAs accepting regular conditions on-demand. The on-the-fly algorithm drastically outperforms the offline algorithm regarding both practical space and time efficiency.

8 Conclusion

We introduce patCPDSs, a subclass of CPDSs with patterns, that are capable of formulating those existing applications that we are aware of, and propose new backward and forward saturation algorithms for solving the reachability problems of the subclass. By directly extending the classic \( \mathcal{P} \)-automata techniques and saturation algorithms of PDSs, the new algorithms avoid an immediate explosion of the resulting systems by taking a detour to solving the reachability problems of PDSs. Furthermore, we identify the subclass of so-called simple patCPDSs for which regular expressions do not contain any intersection and negation operators but basic patterns and the union operator, for which there exist fixed-parameter polynomial algorithms of reachability analysis. We expect that our algorithms would be also amenable to combining with the powerful technique of abstraction refinements and symbolic encodings like BDD (binary decision diagram). We hope that our results would be beneficial to readers who expect a tractable algorithm when formulating their problems as CPDSs.

Acknowledgements This work was partially supported by Shanghai Pujiang Program (Grant No. 17PJ1402200). The author thanks Yuxin Deng for his valuable comments on improving the presentation of the work.

References