Saturation algorithms for model-checking pushdown systems*

Arnaud Carayol
LIGM
Université Paris-Est & CNRS
arnaud.carayol@univ-mlv.fr

Matthew Hague
Department of Computer Science
Royal Holloway University of London
matthew.hague@rhul.ac.uk

We present a survey of the saturation method for model-checking pushdown systems.

1 Introduction

Pushdown systems have, over the past 15 years, been popular with the software verification community. Their stack can be used to model the call stack of a first-order recursive program, with the control state holding valuations of the program's global variables, and stack characters encoding the local variable valuations. As such the control flow of first-order recursive programs (such as C and Java programs) can be accurately modelled [30]. Pushdown systems have played a key role in the automata-theoretic approach to software model checking and considerable progress has been made in the implementation of scalable model checkers of pushdown systems. These tools (e.g. Bebop [3] and Moped [22, 40, 54, 52]) are an essential back-end components of high-profile model checkers such as SLAM [2].

A fundamental result for the model-checking of pushdown systems was established by Büchi in [12]. He showed that the set of stack contents reachable from the initial configuration of a pushdown system form a regular language and hence can be represented by a finite state automaton. The procedure provided by Büchi to compute this automaton from the pushdown system is exponential. In [15], Caucal gave the first polynomial time algorithm to solve this problem. This efficient computation is obtained by a saturation process where transitions are incrementally added to the finite automaton. This technique, which is the topic of this survey, was simplified and adapted to the model-checking setting by Bouajjani *et al.* in [7] and independently by Finkel *et al.* in [23].

The saturation technique allows global model checking of pushdown systems. For example, one may construct a regular representation of all configurations reachable from a given set of initial configurations, or, dually, the set of all configurations that may reach a given set of target configurations. As well as providing direct solutions to simple reachability properties (e.g. can an error state be reached from a designated initial configuration), the representations constructed by global analyses may be reused in a variety of settings. For example, once may perform multiple (and dynamic) queries on the set of reachable states without having to re-run the model checking routine. Additionally, these representations may be combined as part of a larger algorithm or proof. For example, Bouajjani *et al.* provided solutions to the model checking problem for the alternation free μ-calculus by combining the results obtained through multiple global reachability analyses [7].

In this survey, we present the saturation method under its different forms for reachability problems in Section 3. The saturation technique also generalises to the analysis of two-players games played over the configuration graph of a pushdown systems. This extension based on the work of Cachat [13] and Hague and Ong [29] is presented in Section 4. In Section 5, we review the various model-checking tools that

*We thank Didier Caucal and Olivier Serre for helpful discussions. This survey was supported by the Engineering and Physical Sciences Research Council [EP/K009907/1].

Submitted to:
ALF 2014

implement the saturation technique. We conclude in Section 6 by giving an overview of the extensions of the basic model of pushdown system for which the saturation technique has been applied.

2 Preliminaries

2.1 Finite automata

We denote by Σ^* the set of words over the finite alphabet Σ. For $n \geq 0$, we denote by $\Gamma^{\leq n}$ the set of words of length at most n.

A finite automaton $\mathscr{A}$ over the alphabet Σ is a tuple $(\mathbb{S}, \mathscr{I}, \mathscr{F}, \delta)$ where $\mathbb{S}$ is a finite set of states, $\mathscr{I} \subseteq \mathbb{S}$ is the set of initial states, $\mathscr{F} \subseteq \mathbb{S}$ is the set of final states and $\delta \subseteq \mathbb{S} \times \Sigma \times \mathbb{S}$ is the set of transitions. We write $s \xrightarrow[\mathscr{A}]{a} t$ to denote that (s,a,t) is a transition of $\mathscr{A}$. For a word $w \in \Sigma^*$, we write $s \overset{w}{\underset{\mathscr{A}}{\Longrightarrow}} t$ to denote the fact that $\mathscr{A}$ can reach the state t while reading the word w starting from the state s. The language accepted by $\mathscr{A}$ from a state s is

$$\mathscr{L}_s(\mathscr{A}) = \left\{ w \in \Sigma^* \;\middle|\; \exists s_f \in \mathscr{F}. s \overset{w}{\underset{\mathscr{A}}{\Longrightarrow}} s_f \right\}$$

and the language accepted by $\mathscr{A}$ is

$$\mathscr{L}(\mathscr{A}) = \bigcup_{s \in \mathscr{I}} \mathscr{L}_s(\mathscr{A}) \ .$$

2.2 Pushdown system

A pushdown system P is a given by a tuple $(Q, \Gamma, \perp, \Delta)$ where Q is a finite set of control states, Γ is the finite stack alphabet, $\perp \in \Gamma$ is a special bottom of stack symbol and $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma^{\leq 2})$ is the set of transitions. We write $(q,A) \to (p,w)$ for the transition $((q,A),(p,w))$. A configuration is a tuple (q,w) where q is a state in Q and w is a stack content in $(\Gamma \setminus \{\perp\})^* \perp$. In the configuration $c = (q,Aw)$, the pushdown system can apply the transition $(q,A) \to (p,u)$ to go to the configuration $c' = (p,uw)$. As is usual, we assume that transitions of the pushdown system does not pop the bottom of stack symbol or does not push it on the stack (*i.e.* all transitions involving the symbol $\perp$ are of the form $q\perp \to p\perp$ or $q\perp \to p\perp A$ for some $A \in \Gamma \setminus \{\perp\}$). We denote by $\underset{P}{\to}$ (or simply $\to$ if P is clear from the context) the relation on configurations defined by the transitions of P. We denote by $\underset{P}{\Longrightarrow}$ the reflexive and transitive closure of $\underset{P}{\to}$.

3 Reachability problems for pushdown systems

A fundamental result for the model-checking of pushdown systems is the fact that the set of stack contents:

$$\{w \in \Gamma^* \mid \exists q \in Q, (q_0, \perp) \Rightarrow (q,w)\}$$

that are reachable from an arbitrary initial configuration of the system, form a regular set of words over the stack alphabet Γ.

A more elegant formulation of this result can be obtained by extending the notion of regularity to sets of configurations. A set of configurations C is *regular* if for every state $p \in Q$, the set of associated stack contents $\{w \in \Gamma^* \mid (p,w) \in C\}$ is regular. A P-automaton is a slight extension of the standard notion of finite automaton to accept configurations. The only extra assumption is that the set of states of the P-automaton contains the set of states of the pushdown system. Formally, a P-automaton is of the form $(\mathbb{S}, Q, \mathscr{F}, \delta)$ where Q is the set of states of the pushdown system P. A configuration is (p, w) is accepted by $\mathscr{A}$ if w is accepted by $\mathscr{A}$ starting from the state p (*i.e.* $w \in L_p(\mathscr{A})$).

Theorem 1 *[12] The set of configurations of a pushdown system reachable from the initial configuration (*i.e. *the configuration $(q_0, \bot)$ for some arbitrary state q_0) is regular. Moreover a P-automaton accepting it can be effectively constructed from the pushdown system.*

To the authors knowledge, the first proof of this result is due to Büchi in [12]. The formalism used by Büchi is not that of pushdown automata but that of prefix word-rewriting systems (which he calls *regular canonical systems*). These systems syntactically include pushdown automata and conversely can be simulated by pushdown automata. In [24], Greibach formalises the correspondence between the two models and gives a simple proof based on a result on context-free languages proved by Bar-Hillel *et al.* in [4]. Greibach also says that the result (for pushdown automata) was part of the folklore at the time but never appeared in print. Even though effective, these proofs do not provide a polynomial time algorithm[1]. The first polynomial time algorithm is due to Caucal [15, 16] which is based on a saturation procedure of a finite state automaton. The idea behind the saturation method can be traced back to [5]. This method was independently rediscovered and used for model-checking purposes by Bouajjani *et al.* in [7] and Finkel *et al.* in [23].

A more general problem is, given a regular set of configurations C, to compute the set:

$$Post^*_P(C) = \{c' \mid \exists c \in C, c \underset{P}{\Longrightarrow} c'\}$$

of configurations that can be reached from a configuration in C.

The regularity of $Post^*(C)$, for any regular set C, can be derived from Theorem 1. Indeed starting from a pushdown system P and a regular set of configurations C, we can create a new pushdown system P' which using new states builds any configuration in C and afterwards behaves like P. Clearly the set of configurations reachable from the initial configuration of P' coincide with $Post^*_P(C)$ when restricted to the states of P.

As mentioned in the introduction, for model-checking purposes it is often interesting to compute the set of configurations that can reach a given set of *bad* configurations. This leads to consider the set

$$Pre^*_P(C) = \{c' \mid \exists c \in C, c' \Rightarrow c\}$$

of configurations that can reach a configuration in C.

The regularity of $Pre^*(C)$ for any regular set C can be deduced from the regularity of $Post^*(C)$. The intuitive idea is to construct, from P, a new pushdown system P' whose derivation relation is the inverse of that of P. For a transition of the form $qA \rightarrow p$ of P, we add the transitions $pX \rightarrow qAX$ for all symbols $X \in \Gamma$. For a transition $qA \rightarrow pBC$ of P, we add two transition $pB \rightarrow r_{(C,q,A)}$ and $r_{(C,q,A)}C \rightarrow qA$ where $r_{(C,q,A)}$ is a new intermediary control state. For any two configurations c and c' of P, it holds that $c \Rightarrow_P c'$ if and only if $c' \Rightarrow_{P'} c$. Hence $Pre^*_P(C)$ is equal to the restriction of $Post^*_{P'}(C)$ to the states of P and is therefore regular.

The section is structured as follows. We present Büchi's original proof in Section 3.1. In Section 3.2, we present the saturation algorithm to compute $Pre^*(C)$ introduced in [7]. Finally in Section 3.3, we characterise the derivation relation of the pushdown automata using the saturation technique following [15].

[1]We will see Section 3.1 that it can easily be adapted to provide a polynomial time algorithm.

3.1 Büchi's proof

We present a proof of Theorem 1 adapted from [12]. In the original proof, Büchi first reduced the problem to a very simple form of pushdown system where transitions are either of the form $pA \to q$ or $p \to qA$. This model (called *reduced regular systems* by Büchi) is completely symmetric and therefore computing Pre^* or $Post^*$ is essentially the same thing. However to adapt the proof to the formalism used in this article (recall that our formalism does not allow rules of the form $p \to qA$), it is more convenient to work with Pre^* than with $Post^*$.

Given a pushdown system $P = (Q, F, \bot, \Delta)$, we construct a P-automaton accepting $Pre^*_P(\{(q_f, \bot)\})$ where q_f is an arbitrary *final* state of the pushdown system.

The construction is based on the following remark: to reach the configuration $(q_f, \bot)$ from a configuration $(p, Aw\bot)$ it is necessary, at some point, to reach a configuration of the form $(q, w\bot)$ for some state $q \in Q$. Moreover the first time such a configuration is reached, the actions taken by P cannot depend on w since at no point was w exposed at the top of the stack. Hence it must be the case that $pA \Rightarrow q$.

The P-automaton when accepting a stack content $A_1 \dots A_n\bot$ from the state p will guess the states $p_1, \dots, p_n$ such that $pA_1 \underset{P}{\Longrightarrow} p_1$ and $p_iA_{i+1} \underset{P}{\Longrightarrow} p_{i+1}$ for $i \in [0, n-1]$ and will enter a final state upon reading the symbol $\bot$ if $p_n\bot \underset{P}{\Longrightarrow} q_f\bot$.

Consider the P-automaton $\mathscr{A}$ with set of states $Q \cup \{s_\bot\}$ where $s_\bot$ is a new state and the only final state of the automaton. The transitions of the automaton $\mathscr{A}$ are defined as follows:

- $p \xrightarrow{A} q$ if and only if $pA \underset{P}{\Longrightarrow} q$ for all $p, q \in Q$ and $A \in \Gamma \setminus \{\bot\}$,
- $p \xrightarrow{\bot} s_\bot$ if and only if $p\bot \underset{P}{\Longrightarrow} q_f\bot$ for all $p \in Q$.

A simple induction on the length of the stack content shows that $\mathscr{A}$ accepts a stack content $w\bot$ from the state $q \in Q$ if and only if $(q, w\bot)$ belongs to $Pre^*(\{(q_f, \bot)\})$.

To make the construction effective, it remains to compute the relations $pA \Rightarrow q$ and $p\bot \Rightarrow q\bot$ for all states p and $q \in Q$ and stack symbol $A \in \Gamma$. The procedure provided by Büchi is exponential[2]. He first establishes a bound on the height of the stack necessary to build a derivation path witnessing these relations. As the bound is polynomial in the size of the pushdown system, the problem is reduced to a simple reachability problem in a finite graph of exponential size with respect to the size of the pushdown system.

To obtain a polynomial algorithm, it is enough to efficiently compute the relation $Rew = \{(pA, qB) \mid pA \underset{P}{\Longrightarrow} qB\}$. Indeed $pA \underset{P}{\Longrightarrow} q$ if and only if there exists $r \in Q$ and $B \in \Gamma$ such that $pA \underset{P}{\Longrightarrow} rB$ (*i.e.* $(p, A, r, B) \in \text{Rew}$) and $rB \to q$ is a transition of P.

The key idea which is at the heart[3] of the saturation algorithm presented in Section 3.2 is to express Rew as a smallest fixed-point.

The relation Rew is the smallest relation (for the inclusion) in $Q\Gamma \times Q\Gamma$ such that:

- $(pA, pA) \in Rew$ for all $p \in Q$ and $A \in \Gamma$,

[2] In [12], the P-automaton constructed is deterministic (essentially the automaton obtained by applying the power-set construction to the automaton presented here). With the added constraint of determinism, it not possible to obtain a polynomial algorithm as the smallest deterministic automaton is in general exponential in the size of the pushdown system. To convince oneself, it is enough to consider a pushdown system that simulates a non-deterministic finite state automaton (NFA) by popping its stack until the bottom of the stack is reached and when the bottom of the stack is reached goes to the state q_f if the NFA has reached a final state.

[3] We will see that the algorithm presented in Section 3.2 performs a fixed-point computation for the relation $\{(pA, q) \mid pA \underset{P}{\Longrightarrow} q\}$.

- $(pA, qB) \in \text{Rew}$ if $pA \to qB$ is a transition of P,
- $(pA, qC) \in \text{Rew}$ if $(pA, rB) \in \text{Rew}$ and $(rB, qC) \in \text{Rew}$,
- $(pA, qC) \in \text{Rew}$ if $pA \to rBC$ is a transition of P and there exists $t \in Q$ and $D \in \Gamma$ such that $(rB, tD) \in \text{Rew}$ and $tD \to q$ is a transition of P.

The property (1) expresses that Rew is reflexive and (3) that it is transitive. Property (2) ensures that Rew contains the relevant transitions of P. Property (4) describes the case when $pA \underset{P}{\Longrightarrow} qC$ is obtained by a sequence of the form $pA \underset{P}{\to} rBC \underset{P}{\Longrightarrow} qC$ where $rB \underset{P}{\Longrightarrow} q$.

Using the Knaster-Tarski theorem, we can compute Rew as the limit of an increasing sequence of relations $(\text{Rew}_i)_{i\geq 0}$ over $Q \times \Gamma$. The relation Rew_0 contains the elements satisfying property (1) and (2). The relation Rew_{i+1} is obtained from Rew_i by adding all the elements that can be derived by property (3) or (4) in Rew_i. The sequence $(\text{Rew}_i)_{i\geq 0)}$ is increasing for the inclusion and its limit (*i.e.* the first set such that $\text{Rew}_{i+1} = \text{Rew}_i$) is equal to Rew. As at least one element is added at each step before the limit is reached, the limit is reached in at most $|Q|^2|\Gamma|^2$ steps. Furthermore as the computation of Rew_{i+1} from Rew_i can be done in polynomial time with respect to the size of P, the resulting algorithm is polynomial. However the exact complexity is not as good as the algorithm presented in Section 3.2.

3.2 Saturation algorithm of [7]

In [7], Bouajjani *et al.* present an algorithm that given a pushdown system $P = (Q, \Gamma, \perp, \Delta)$ and a P-automaton $\mathscr{A} = (\mathbb{S}, Q, \delta, \mathscr{F})$, constructs a new P-automaton $\mathscr{B}$ accepting $Pre^*_P(\mathscr{L}(\mathscr{A}))$. The only requirement on $\mathscr{A}$ is that no transition in δ goes back to a state in Q[4]. This restriction also implies that none of the states in Q are final.

The algorithm proceeds by adding transitions to $\mathscr{A}$ following a unique rule until no new transition can be added. The resulting P-automaton $\mathscr{B}$ accepts the set of configurations $Pre^*_P(\mathscr{L}(\mathscr{A}))$.
More precisely, the algorithm constructs a finite sequence $(\mathscr{A}_i)_{i\in[0,N]}$ of P-automata. The first P-automaton $\mathscr{A}_0$ is the automaton $\mathscr{A}$. All the P-automata $\mathscr{A}_i$ are of the form $(\mathbb{S}, Q, \mathscr{F}, \delta_i)$, meaning that they only differ by their set of transitions. The construction guaranties that for all $i \in [0, N-1]$, $\delta_i \subseteq \delta_{i+1}$ and terminates when $\delta_{i+1} = \delta_i$. As at least one transition is added at each step, the algorithm terminates in at most $|Q|^2|\Gamma|$ steps.

The set of transitions δ_{i+1} is obtained by adding to δ_i, the transition:

$$p \xrightarrow{A} s \text{ if } q \underset{\mathscr{A}_i}{\overset{w}{\Longrightarrow}} s \text{ and } pA \to qw \text{ is a transition of } P.$$

Note that only transitions starting with a state of Q are added by the algorithm. In particular, the language accepted the automaton $\mathscr{A}_i$ from a state in $\mathbb{S} \setminus Q$ never changes.[5].

The construction of δ_{i+1} from δ_i ensures that the configurations that can reach in one step a configuration in $\mathscr{L}(\mathscr{A}_i)$ belong to $\mathscr{L}(\mathscr{A}_{i+1})$. Consider two configurations $c = (p, Au)$ and $c' = (q, wu)$ such that $pA \to qw$ is a transition of P (and hence $c \underset{P}{\to} c'$). Now assume that c' belongs to $\mathscr{L}(\mathscr{A}_i)$. This means that for some state $s \in \mathbb{S}$ and some final state $s_f \in \mathscr{F}$, $q \underset{\mathscr{A}_i}{\overset{w}{\Longrightarrow}} s \underset{\mathscr{A}_i}{\overset{u}{\Longrightarrow}} s_f$. The rule of construction of δ_{i+1} ensures that $p \xrightarrow{A} s$ is a transition of $\mathscr{A}_{i+1}$. Hence $p \underset{\mathscr{A}_{i+1}}{\overset{A}{\Longrightarrow}} s \underset{\mathscr{A}_{i+1}}{\overset{u}{\Longrightarrow}} s_f$ and the configuration $c = (p, Au)$ is accepted by $\mathscr{A}_{i+1}$. As $\mathscr{B}$ is the limit of the saturation process (*i.e.* $\mathscr{B} = \mathscr{A}_{N-1} = \mathscr{A}_N$), $\mathscr{L}(\mathscr{B})$ is closed under taking

[4]This requirement is easily met by adding a copy of each state in Q if necessary. This restriction is required to ensure that the first invariant maintained by the algorithm holds initially.

[5]Recall that initially the states in Q are not the target of any transition

the immediate predecessor for the relation $\underset{P}{\rightarrow}$ (*i.e.* if $c' \in \mathscr{L}(\mathscr{B})$ and $c \underset{P}{\rightarrow} c'$ then $c \in \mathscr{L}(\mathscr{B})$). As $\mathscr{L}(\mathscr{B})$ includes $\mathscr{L}(\mathscr{A})$, it follows that $Pre^*_P(\mathscr{L}(\mathscr{A})) \subseteq \mathscr{L}(\mathscr{B})$.

The proof of the converse inclusion requires a more careful analysis. The algorithm maintains two invariants on the transitions in δ_i. For all $i \in [0,N]$, the presence of a transition $p \xrightarrow{A} s$ in δ_i guaranties that:

1. $pA \underset{P}{\Longrightarrow} s$ if s belongs to Q.
2. the configuration (p, Au) belongs to $Pre^*(\mathscr{L}(\mathscr{A}))$ for any $u \in \mathscr{L}_s(\mathscr{A}_i) = \mathscr{L}_s(\mathscr{A})$ if s belongs to $\mathbb{S} \setminus Q$.

From these invariants, it follows that for all $i \geq 0$, $\mathscr{L}(\mathscr{A}_i) \subseteq Pre^*_P(\mathscr{L}(\mathscr{A}))$. In particular, $\mathscr{L}(\mathscr{B}) \subseteq Pre^*_P(\mathscr{L}(\mathscr{A}))$.

Remark 1 *As indicated by the first invariant, if we restrict our attention to transitions with both source and target in Q, this algorithm is performing a fixed-point computation for the relation* $\underset{P}{\Longrightarrow}$ *restricted to* $(Q \times \Gamma) \times (Q \times \{\varepsilon\})$. *Indeed this relation can be characterised as the smallest relation (for the inclusion)* $\mathscr{R}$ *such that:*

1. $pA \mathscr{R} q$ *if* $pA \rightarrow q$ *belongs to* Δ,
2. $pA \mathscr{R} q$ *if* $rB \mathscr{R} q$ *and* $pA \rightarrow rB$ *belongs to* Δ,
3. $pA\mathscr{R}q$ *if* $pA \rightarrow rBC$ *belongs to* Δ *and for some state* $s \in Q$, $rB\mathscr{R}s$ *and* $sC\mathscr{R}q$.

In fact, the algorithm performs the computation of the smallest such relation following the procedure given by Knaster-Tarski theorem.

A naive implementation of this algorithm yields a complexity in $\mathscr{O}(|P|^2|\mathscr{A}|^3)$. However a more efficient implementation presented in [20] lowers the complexity to $\mathscr{O}(|Q|^2|\Delta|)$.

In [20], an adaptation of the algorithm for computing Pre^* is given to compute $Post^*$. The algorithm is slightly less elegant as it requires the addition of new states before the saturation process. In fact, it is very similar to first applying the transformation to invert the pushdown system presented at the beginning of this section and then applying the algorithm to compute Pre^*.

In [41], Schwoon shows how to use the saturation algorithm to construct for any configuration c accepted by $\mathscr{B}$ a derivation path to some configuration in $\mathscr{L}(\mathscr{A})$.

3.3 Derivation relation of a pushdown system

In this section, we will see that the saturation method can be adapted to characterise the derivation relation of a pushdown system. Let us fix a pushdown system[6] $P = (Q, \Gamma, \Delta)$, an initial state q_0 and a final state q_f. We aim at giving an effective characterisation of the following relation between stacks:

$$\mathrm{Deriv}_P = \{(u,v) \in \Gamma^* \mid (q_0, u) \underset{P}{\Longrightarrow} (q_f, v)\}.$$

In [15], Caucal showed that $\mathrm{Deriv}_P \subseteq \Gamma^* \times \Gamma^*$ is a rational relation, *i.e.* it is accepted by a finite state automaton with output (also called a transducer).

The proof presented here is based on [17] but similar ideas can be found in [39, 23]. The idea of the proof is to use symbols to represent the actions of the pushdown system on the stack: one symbol for

[6]To simplify the presentation, we do not take the bottom of stack symbol into account.

pushing a given symbol and one symbol for popping it. The pushdown system is transformed into a finite state automaton that instead of performing the actions on the stack outputs the symbols that represent these actions (see Section 3.3.1). This finite state automaton is then transformed using a saturation algorithm so that it erases sequences of actions corresponding to pushing a symbol and then immediately popping it (see Section 3.3.2). From this *reduced* language, the relation Deriv_P is easily characterised (see Section 3.3.3).

3.3.1 Sequences of stacks actions

For every symbol $A \in \Gamma$, we introduce two symbols:

- A_+ which represents the action of pushing the symbol A on top of the stack,
- and A_- which represents the action of popping the symbol A from the top of the stack.

We denote by Γ_+ the set $\{A_+ \mid A \in \Gamma\}$ of *push* actions, by Γ_- the set $\{A_- \mid A \in \Gamma\}$ of *pop* actions and by $\overline{\Gamma}$ the set $\Gamma_+ \cup \Gamma_-$ of all action symbols.

Intuitively a sequence $\alpha = \alpha_1 \dots \alpha_n \in \overline{\Gamma}^*$ is interpreted as performing the action α_1, followed by the action α_2 and so on. For instance, the effect on the stack of the transition $pA \to qBC$ is represented by the word $A_-C_+B_+$. First the automaton removes the A from the top of the stack and then pushes C and then B.

For two stacks u and $v \in \Gamma^*$, we write $u \overset{\alpha}{\rightsquigarrow} v$ if u can be transformed into v by the sequence of actions α. For instance, we have $ABB \overset{\alpha}{\rightsquigarrow} DCB$ for the α sequence $A_-B_-C_+D_+$. Note that some sequences of actions such as B_+C_- cannot be applied to any stack. We say that such sequences α are *non-productive*, *i.e.* there are no u and $v \in \Gamma^*$ such that $u \overset{\alpha}{\rightsquigarrow} v$.

From the pushdown system P, we can construct a regular set of action sequences denoted Behaviour_P which contains all the sequences (even the non-productive ones) that can be performed by P when starting in state q_0 and ending in state q_f. Consider for instance the finite state automaton[7] $(Q, \{q_0\}, \{q_f\}, \delta)$ where the set of transitions δ is given by:

$$\begin{cases} p \xrightarrow{A_-C_+B_+} q \in \delta \quad \text{if} \quad pA \to qBC \in \Delta \\ p \xrightarrow{A_-B_+} q \in \delta \quad \text{if} \quad pA \to qB \in \Delta \\ p \xrightarrow{A_-} q \in \delta \quad \text{if} \quad pA \to q \in \Delta \end{cases}$$

It is clear that Behaviour_P characterises Deriv_P in the following sense:

$$(u,v) \in \text{Deriv}_P \quad \text{if and only if} \quad u \overset{\alpha}{\rightsquigarrow} v \ \text{ for some } \alpha \in \text{Behaviour}_P.$$

However this representation of Deriv_P is not very helpful in its current form. For instance, Behaviour_P can contain non-productive sequences or sequences such as $A_-B_+A_+A_-C_+C_-$ which is equivalent to the more informative sequence A_-B_+.

[7]The finite state automaton does not strictly conform to the definition we gave in Section 2 as its transitions are labelled by words and not single letters. This can be easily avoided at the cost of adding intermediate states.

3.3.2 Reducing sequences of actions

To simplify Behaviour$_P$, we first erase all factors of the form A_+A_- for $A \in \Gamma$. These factors can safely be omitted as they do not affect the stack: the symbol is pushed then immediately popped. A sequence that does not contain any such factors is called *reduced*.

To perform this erasure, we introduce the relation $\mapsto$ which relates a stack $u \in \Gamma^*$ and a stack $v \in \Gamma^*$ if v can be obtained by erasing a factor A_+A_- from u (*i.e.* $u = u_1A_+A_-u_2$ and $v = u_1u_2$). Clearly, if $\alpha \mapsto \beta$ then the sequences α and β are equivalent with respect to their actions on the stack :

$$\text{for } u,v \in \Gamma^*,\ u \overset{\alpha}{\rightsquigarrow} v \text{ if and only if } u \overset{\beta}{\rightsquigarrow} v.$$

As the rewriting relation $\mapsto$ is confluent and decreases the length of the sequence, every sequence α can be iteratively rewritten by $\mapsto$ into a reduced sequence denoted $\mathrm{Red}(\alpha)$. For instance the reduced sequence associated to $B_-A_+A_+A_-A_-C_+$ is B_-C_+ as $B_-A_+A_+A_-A_-C_+ \mapsto B_-A_+A_-C_+ \mapsto B_-C_+$.

In [5], Benois showed[8] that the set of reduced sequences corresponding to a regular set of sequences is again regular.

Theorem 2 *[5, 6] For any regular set R of action sequences, the corresponding set of reduced action sequences:*

$$\mathrm{Red}(R) = \{\mathrm{Red}(\alpha) \mid \alpha \in R\}$$

is regular. Moreover given a finite automaton $\mathscr{A}$ accepting R, an automaton accepting $\mathrm{Red}(R)$ *can be constructed in $\mathscr{O}(|\mathscr{A}|^3)$.*

The proof of this theorem is the essence of the saturation method. Starting with the automaton $\mathscr{A}$, ε-transitions are added until no new ε-transition can be added. The ε-transitions are added according to the following rule. We add an ε-transition from a state p to a state q if it is possible to reach q from p reading a word of form $A_+\varepsilon^*A_-$. It can be shown that the resulting saturated automaton accepts the language:

$$\{\beta \in \overline{\Gamma}^* \mid \alpha \mapsto^* \beta \quad \text{for some } \alpha \in R\}.$$

The construction is concluded by taking the ε-closure of the saturated automaton and restricting the language to the set of reduced sequences (which is a regular language as it is the complement of the language $\cup_{A\in\Gamma}\overline{\Gamma}^*A_+A_-\overline{\Gamma}^*$). A careful implementation of the procedure presented in [6] gives an algorithm in $\mathscr{O}(|\mathscr{A}|^3)$.

3.3.3 Characterisation of Deriv$_P$

One of the advantages of working with Red(Behaviour$_P$) is that we can easily remove non-productive sequences. Indeed a reduced sequence is non-productive if and only if it contains a factor of the form A_+B_- for $A \neq B \in \Gamma$.

We can hence compute the regular language:

$$RP_P = \mathrm{Red}(\mathrm{Behaviour}_P) \cap \left(\overline{\Gamma}^* \setminus \bigcup_{A\neq B\in\Gamma} \overline{\Gamma}^*A_+B_-\overline{\Gamma}^*\right)$$

which is composed of the reduced and productive action sequences characterising Deriv$_P$.

[8]Benois consider the erasure of all factor of the form A_-A_+ as well as A_+A_- but the proof is identical.

The language RP_P does not contain any factor in $\Gamma_+\Gamma_-$ and is hence included in $\Gamma_-^*\Gamma_+^*$. We can express it as a finite union:

$$\bigcup_{i\in[1,N]} X_iY_i$$

where for all $i \in [1,N]$, X_i is a regular language in Γ_-^* and Y_i is a regular language in Γ_+^*.

Let us denote by U_i the regular set $\{A^1 \cdots A^n \in \Gamma^* | \mid A_-^1 \cdots A_-^n \in X_i\}$ of words in Γ^* that can be popped by a sequence in X_i and by V_i the regular set $\{A^1 \cdots A^n \in \Gamma^* | \mid A_+^n \cdots A_+^1 \in Y_i\}$ of words in Γ^* that can be pushed by a sequence in Y_i.

The relation Deriv_P can be characterised as follows: a pair (w_1, w_2) belongs to Deriv_P, if for some $i \in [1,N]$, w_1 can be written as uw with $u \in U_i$ and w_2 can be written as vw for some $v \in V_i$. In other terms, the relation Deriv_P can be written as a finite union of relations that remove a prefix of the stack belonging to a certain regular language and replace any word in another regular language. As these relations are easily accepted by finite transducer, so is Deriv_P. Combining all the steps, we obtain a polynomial time algorithm for computing a transducer accepting Deriv_P from P.

4 Winning regions of pushdown games

The saturation technique also generalises to the analysis of pushdown games with two players: Éloise and Abelard. The two players may, for example, model a program (Éloise) interacting with the environment (Abelard). While the program can control its next move based on its internal state, it cannot control the results of requesting external input. Hence, the external input is decided by the second player.

A pushdown game may be used to analyse various types of properties. We will consider three, increasingly expressive, types of properties here: reachability, Büchi and parity. We will begin by defining games with generic winning conditions and then consider the instantiations of this generic framework for each winning condition in turn. We will simultaneously discuss the saturation algorithm for each of these properties and show how they build upon each other.

The saturation algorithm was first extended to pushdown reachability games by Bouajjani *et al.* [7]. Their algorithm was extended to the case of Büchi games by Cachat [13] and then to parity games by Hague and Ong [29]. Our presentation will follow that of Hague and Ong since it provides the most general algorithm, though we remark that all the essential ideas of the algorithm were in place by the introduction of the Büchi algorithm. The main contribution of Hague and Ong was a proof framework that simplified the technical arguments by Bouajjani *et al.* and Cachat and allowed the full parity case to go through.

4.1 Preliminaries

4.1.1 Pushdown games

We can obtain a two-player game from a pushdown system P by the addition of two components: a partition of the configurations of P into positions controlled by Éloise and positions controlled by Abelard; and the definition of a winning condition that determines the winner of any given play of the game.

In the following, for technical convenience, we will assume for each $q \in Q$ and $A \in \Gamma$ there exists some $(q,A) \to (p,w) \in \Delta$. Together with the bottom-of-stack symbol, this condition ensures that from a configuration $(q, w\bot)$ it is not possible for the system to become stuck; that is, reach a configuration with no successor.

A two-player pushdown game is a tuple $P = (Q, \Gamma, \bot, \Delta, W)$ such that $(Q, \Gamma, \bot, \Delta)$ defines a pushdown system, Q is partitioned $Q = Q_E \uplus Q_A$ into Éloise and Abelard positions respectively, and W is a set of infinite sequences of configurations of P.

A play of a pushdown game is an infinite sequence $(q_0, w_0), (q_1, w_1), \ldots$ where (q_0, w_0) is some starting configuration and (q_{i+1}, w_{i+1}) is obtained from (q_i, w_i) via some transition $(q_i, A) \rightarrow (q_{i+1}, w) \in \Delta$. In the case where $q_i \in Q_E$ it is Éloise who chooses the transition to apply, otherwise Abelard chooses the transition.

The winner of an infinite play $(q_0, w_0), (q_1, w_1), \ldots$ is Éloise if $(q_0, w_0), (q_1, w_1), \ldots \in W$; otherwise, Abelard wins the play. The winning region $\mathscr{W}$ of a pushdown game is the set of all configurations from which Éloise can always win all plays, regardless of the transitions chosen by Abelard.

4.1.2 Alternating automata

To extend the saturation algorithm to compute the winning region of a pushdown game, we augment the automata used to recognise sets of configurations with alternation. Bouajjani *et al.* first used alternating automata to analyse pushdown reachability games via saturation [7], however, they used the equivalent formalism of *alternating pushdown systems* rather than pushdown games. An alternating automaton is a tuple $\mathscr{A} = (\mathbb{S}, \Gamma, \mathscr{F}, \delta)$ where $\mathbb{S}$ is a finite set of states, Γ is a finite alphabet, $\mathscr{F} \subseteq \mathbb{S}$ is the set of accepting states, and $\delta \subseteq \mathbb{S} \times \Gamma \times 2^{\mathbb{S}}$ is a transition relation. Note that we do not specify a set of initial states. This is because it is more convenient to present the following results in terms of the stacks accepted from particular states, rather than fixing a set of initial states.

Whereas a transition $s \xrightarrow{A} t$ of a non-deterministic automaton requires the remainder of the word to be accepted from t, a transition $s \xrightarrow{A} S$ of an alternating automaton requires that the remainder of the word is accepted from all states $s' \in S$. It is this "for all" condition that captures the fact that Éloise must be able to win for all moves Abelard may make.

More formally, a run over a word $A_1 \ldots A_n \in \Gamma^*$ from a state s_0 is a sequence

$$S_1 \xrightarrow{A_1} \cdots \xrightarrow{A_n} S_{n+1}$$

where each S_i is a set of states such that $S_1 = \{s_0\}$, and for each $1 \leq i \leq n$ we have

$$\forall s \in S_i . \exists s \xrightarrow{A_i} S \in \delta \wedge S \subseteq S_{i+1} \ .$$

The run is accepting if $S_{n+1} \subseteq \mathscr{F}$. Thus, for a given state s, we define $\mathscr{L}_s(\mathscr{A})$ to be the set of words over which there is an accepting run of $\mathscr{A}$ from $\{s\}$.

When S_i is a singleton set, we will often omit the set notation. For example, the run above could be written

$$s_0 \xrightarrow{A_1} \cdots \xrightarrow{A_n} S_{n+1} \ .$$

Further more, when $w = A_1 \ldots A_n$ we will write $s \xrightarrow{w} S$ as shorthand for a run from s to S.

4.2 Pushdown reachability games

One of the simplest winning conditions for a game is the reachability condition. Given a target set of configurations C, the reachability condition states that Éloise wins the game from a given configuration if she can force all plays starting at that configuration to some configuration in C.

That is, a pushdown reachability game is a tuple $(Q,\Gamma,\bot,\Delta,C)$ such that (Q,Γ,Δ,W) is a pushdown game where

$$W = \{c_0, c_1, \dots \mid \exists i. c_i \in C\}$$

is the set of all sequences of configurations containing some configuration in C.

4.2.1 Characterising the winning region

In the sequel we will need to combine least and greatest fixed points. We will use μ to denote the least fixed point operator, and ν to denote the greatest fixed point operator.

In the simple case of reachability for a pushdown system P and set of target configurations C we can characterise the winning region $\mathscr{W} = Pre^*_P(C)$ as

$$\mu Z. C \cup Pre_P(Z)$$

where

$$Pre_P(Z) = \left\{ (p,w) \;\middle|\; \begin{array}{lll} p \in Q_E & \Rightarrow & \exists (p,w) \to c.\, c \in Z \quad \wedge \\ p \in Q_A & \Rightarrow & \forall (p,w) \to c.\, c \in Z \end{array} \right\} .$$

That is, to appear in $\mathscr{W}$ for a configuration belonging to Éloise, it must be possible for her to choose a transition that progresses towards C. For configurations belonging to Abelard, it must be the case that he cannot help but choose a transition that progresses towards C.

4.2.2 Computing the winning region

Fix a pushdown reachability game $P = (Q,\Gamma,\Delta,C)$. We will show how to construct an automaton $\mathscr{B}$ whose state set includes the state p for all $p \in Q$ and $w \in \mathscr{L}_p(\mathscr{B})$ iff $(p,w) \in \mathscr{W}$.

Computing Éloise's winning region is a direct extension of the saturation algorithm for $Pre^*_P(C)$ in the non-game setting. We assume C is a regular set of configurations represented by an alternating automaton $\mathscr{A} = (\mathbb{S},\Gamma,\delta,\mathscr{F})$ such that $Q \subseteq \mathbb{S}$ and there are no-incoming transitions to any state in Q.

The saturation algorithm constructs the automaton $\mathscr{B}$ that is the least fixed point of the sequence of automata $\mathscr{A}_0, \mathscr{A}_1, \dots$ where $\mathscr{A}_0 = \mathscr{A} = (\mathbb{S},\Gamma,\delta_0,\mathscr{F})$ and $\mathscr{A}_{i+1} = (\mathbb{S},\Gamma,\delta_{i+1},\mathscr{F})$ where δ_{i+1} is the smallest set of transitions such that

1. $\delta_i \subseteq \delta_{i+1}$, and
2. for each $q \in Q_E$, if $(q,A) \to (p,w) \in \Delta$ and $p \xrightarrow{w} S$ is a run of $\mathscr{A}_i$, then
$$q \xrightarrow{a} S \in \delta_{i+1}$$
and
3. for each $q \in Q_A$ and $A \in \Gamma$ and $S \subseteq \mathbb{S}$ such that for all
$$(q,A) \to (p,w) \in \Delta$$
there exists a run $p \xrightarrow{w} S'$ of $\mathscr{A}_i$ with $S' \subseteq S$, we have
$$q \xrightarrow{a} S \in \delta_{i+1} \ .$$

One can prove that $(p,w) \in \mathscr{W}$ iff $w \in \mathscr{L}_p(\mathscr{B})$. Thus we obtain regularity of the winning region. Since the maximum number of transitions of an alternating automaton is exponential in the number of states (and we do not add any new states), we have that $\mathscr{B}$ is constructible in exponential time.

Theorem 3 *The winning region of a pushdown reachability game is regular and constructible in exponential time.*

4.2.3 Winning strategies

Cachat has given two realisations of Éloise's winning strategy in a pushdown reachability game from a configuration in her winning region [13] . The first is a positional strategy that requires space linear in the size of the stack to compute. That is, he gives an algorithm that reads the stack and prescribes the next move that Éloise should make in order to win the game. The algorithm assigns costs to accepting runs of $\mathscr{B}$ for configurations in $\mathscr{W}$ by summing costs assigned to individual transitions.

Alternatively, Cachat presents a strategy that can be implemented by a pushdown automaton that tracks the moves of Abelard and recommends moves to Éloise. Since the automaton tracks the game, the strategy is not positional. However, the prescription of the next move requires only constant time.

In his PhD. thesis [14], Cachat also argues that similar strategies can be computed for Abelard for positions in his winning region.

4.3 Pushdown Büchi games

Plays of a game are infinite sequences. The reachability condition only depends on finite prefixes of these plays, hence games are won within a finite number of moves. This prevents the specification of liveness properties such as "every request is followed by an acknowledgment". Since it is not possible to know when to "stop waiting" for an acknowledgment to arrive, it is not possible to specify such conditions as simple reachability properties.

Büchi conditions allow liveness properties to be defined since deciding the winner of a particular play can take the whole infinite sequence into account. We define a pushdown Büchi game as a tuple $(Q,\Gamma,\perp,\Delta,F)$ – where $F \subseteq Q$ is a set of target control states – which defines a pushdown game $(Q,\Gamma,\perp,\Delta,W)$ with

$$W = \left\{(p_0,w_0),(p_1,w_1),\dots \;\middle|\; \forall i.\exists j \geq i.p_j \in F\right\} .$$

That is, Éloise wins the play if there is some control state in F that is visited infinitely often.

Cachat generalised the saturation method to construct the winning region of a pushdown Büchi game [13] by introducing the nesting of fixed point computations and projection described below.

To characterise the winning region of a pushdown Büchi game, a single least fixed point computation no longer suffices. Intuitively this is because satisfying the Büchi condition amounts to repeatedly satisfying a reachability condition; that is, repeatedly reaching a control state in F. We will begin by giving the characterisation, and then decoding it in the following paragraphs. By abuse of notation, we will write F to also denote the set of configurations $\{(p,w) \mid p \in F\}$ and $\overline{F}$ to denote its complement. The winning region of Éloise can be defined as

$$\nu Z_0.\mu Z_1.\left(F \cap Pre_P(Z_0)\right) \cup \left(\overline{F} \cap Pre_P(Z_1)\right) .$$

There are two pre-steps in the formula: $Pre_P(Z_0)$ and $Pre_P(Z_1)$. When a configuration is in F then we require that Éloise can force the next step of play to stay within Z_0. When the configuration is not in F we require that Éloise can force play to stay within Z_1.

To understand the role of the different fixed points, imagine a game where there is only one move from some configuration (p,w)

$$(p,w) \rightarrow (p,w) .$$

In the case where $p \in F$ it will be the case that (p,w) appears in the greatest fixed point Z_0. This is because greatest fixed points can be "self-supporting": if we include (p,w) in an approximation of Z_0, then it will appear in the next approximation of Z_0 by virtue of the fact that it was in the old valuation.

In the other case, when $p \notin F$, we would require (p,w) to appear in the least fixed point Z_1. However, since the least fixed point is the smallest possible fixed point, its members cannot be self-supporting. That is, if we took (p,w) out of our approximation, the next approximation would not include (p,w): there is nothing external compelling (p,w) to be in the least fixed point. This is why a reachability property is a least fixed point: it must contain only the configurations that eventually reach a target configuration – it cannot put off satisfying this obligation for an infinite number of steps.

In terms of Büchi games this difference makes sense: a play that repeatedly visits only the configuration (p,w) is only winning if $p \in F$. If $p \notin F$ then a configuration can only be winning if it eventually (after a finite number of steps) moves to a configuration that has a control state in F. Thus, the least fixed point represents configurations that must eventually reach a "good" configuration, while the greatest fixed point represents good configurations that are able to support themselves.

4.3.1 Computing the winning region

Automaton representation of multiple fixed points The saturation method for reachability properties computed a single fixed point with a single fixed point variable. We can think of the successive automata $\mathscr{A}_0, \mathscr{A}_1, \ldots$ as successive approximations of the value of Z. The final automaton computed gives the value of Z that is the solution to

$$\mu Z.C \cup Pre_P(Z) \ .$$

In the case of Büchi games, there are two nested fixed point computations over the variables Z_0 and Z_1. The winning region is the greatest fixed point for Z_0. However, in order to compute this fixed point we also have to compute the least fixed point for Z_1. Hence, we will need an automaton that can represent two different sets of configurations: the approximation of Z_0 as well as the approximation of Z_1. Thus, instead of having a state p of the alternating automaton for each control state p, we will have two states p^0 and p^1. A configuration (p,w) appears in the current approximation of Z_0 if it is accepted from p^0, and it appears in the current approximation of Z_1 if it is accepted from p^1. We will also use control states of the form p^2 to hold intermediate values of the computation.

Finally, the automata we build will have two additional states (these will be the only states that are not of the form p^α for some α). There will be one state $s_\perp$ that will be the only accepting state. Since all stacks finish with the bottom-of-stack symbol $\perp$, this state will have no outgoing transitions, and all incoming transitions will be of the form $s \xrightarrow{\perp} \{s_\perp\}$. No other transitions in the automaton will be labelled $\perp$.

The other additional state is s^* from which all stacks are accepted. This state has the outgoing transitions $s^* \xrightarrow{A} \{s^*\}$ for all $A \in \Gamma$ with $A \neq \perp$, and $s^* \xrightarrow{\perp} \{s_\perp\}$.

Evaluation strategy The saturation method computes fixed points following Knaster-Tarski theorem. That is, to compute a least fixed point, it begins with the smallest potential value (the set of target configurations C in the case of reachability properties, and the empty set in the case of Büchi properties). It then adds configurations to this set (by adding new transitions) that also necessarily appear in the least fixed point. This process is repeated until nothing more needs to be added – at which point the least fixed point has been calculated.

To compute a greatest fixed point Z_0 we follow the dual strategy. We begin with the largest possible value, which is the set of all configurations, which we will represent by states p^0 with all possible outgoing transitions. Next, the least fixed point Z_1 is calculated given the initial approximation of Z_0. Once the value of Z_1 is known, it becomes our new approximation of Z_0. Notice that this approximation is necessarily smaller than the initial attempt (both in terms of configurations accepted and transitions present). We then recalculate the least fixed point for Z_1 with the new smaller value of Z_0. In this way, starting from the largest possible value for Z_0 we successively shrink its value until a fixed point is found. This fixed point will be the greatest fixed point.

Projection When computing the greatest fixed point for Z_0 we repeatedly compute a least fixed point for Z_1. Each fixed point for Z_1 becomes the new approximation of Z_0. Hence, during our algorithm we need a method of assigning the value of Z_1 to Z_0. We call this manipulation of transitions *projection*.

Suppose the only outgoing transition from p^1 is

$$p^1 \xrightarrow{A} \{q^1, p^0\}$$

and we want to assign the new value of p^0. To do this we simply remove all transitions from p^0 (the old value) and introduce the transition

$$p^0 \xrightarrow{A} \{q^0, p^0\} \ .$$

There are several things to notice about this new transition. The first is that it emanates from p^0 rather than p^1. Next, we have changed the target state q^1 to q^0. This is because we are renaming all the states annotated with 1 to be annotated with 0. Finally, notice that we have not changed the target state p^0.

By leaving p^0 we are no longer simply transferring the value of Z_1 to Z_0 since we are changing the outgoing transitions from p^0. It is provable that this change in value is benign with respect to the fixed point of Z_0: since p^0 should accept all configurations (p,w) in the fixed point for Z_0, the fact that any run that reaches p^0 may accept additional configurations coming from the new value of p^0 rather than the old simply means that we are accelerating the computation of the fixed point.

For example, suppose we had a pushdown Büchi game with $p \in F \cap Q_E$ and an automaton with the transitions

$$p^1 \xrightarrow{A} \{p^0\} \text{ and } p^1 \xrightarrow{\perp} \{s_\perp\} \text{ and } p^0 \xrightarrow{\perp} \{s_\perp\}$$

and the pushdown game contains (amongst others) the rule $(p,A) \to (p,\varepsilon)$. In particular we accept the configuration $(p, A\perp)$ from p^1, and we do so because we can pop the A to reach $(p,\perp)$ (from which we suppose Éloise can win the game). After projection, we will have the transitions

$$p^0 \xrightarrow{A} \{p^0\} \text{ and } p^0 \xrightarrow{\perp} \{s_\perp\} \ .$$

Notice we now have a loop from p^0 enabling any configuration of the form $(p, A^*\perp)$ to be accepted from p^0. Thus we have increased the valuation during projection. However, this is benign because, by repeated applications of $(p,A) \to (p,\varepsilon)$ Éloise can reach $(p,\perp)$ and win the game. Thus, the projection has collapsed an unbounded sequence of moves into a single transition.

To calculate the fixed point for Z_1 we begin with the empty set as an initial approximation. Then we compute the new approximation for Z_1. While computing this approximation we will use states of the form p^2 to store the new value. Thus, to assign the new approximation to Z_1 we simply perform projection from the states p^2 to p^1 in the same way that we projected when assigning Z_1 to Z_0.

We thus define a projection function on states

$$\pi_{\alpha,\beta}(s) = \begin{cases} s & s = s^* \vee s = s_\perp \\ s & s = p^\gamma \wedge \gamma \neq \alpha \\ p^\beta & s = p^\alpha \end{cases}$$

which generalises naturally to a function on sets of states $\pi_{\alpha,\beta}(S) = \left\{\pi_{\alpha,\beta}(s) \mid s \in S\right\}$.

Algorithm Fix a pushdown Büchi game $P = (Q, \Gamma, \bot, \Delta, F)$. We begin our presentation of the algorithm by presenting a simple function for performing the projections described above. The function Proj($\mathscr{A}$, α, β) projects the value of the states p^α to p^β and deletes all the states p^α.

```
function Proj($\mathscr{A}$, $\alpha$, $\beta$)
    $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
    $\mathbb{S}' \leftarrow \mathbb{S} \setminus \{p^\alpha \mid p \in Q\}$
    $\delta' \leftarrow \begin{array}{l} \left\{ s \xrightarrow{A} S \in \delta \;\middle|\; \forall p \in Q.s \neq p^\alpha \wedge s \neq p^\beta \right\} \cup \\ \left\{ p^\beta \xrightarrow{A} \pi_{\alpha,\beta}(S) \;\middle|\; p^\alpha \xrightarrow{A} S \in \delta \right\} \end{array}$
    return $(\mathbb{S}', \Gamma, \delta', \mathscr{F})$
end function
```

The main algorithm contains two nested fixed point computations: the outer for Z_0 and the inner for Z_1. The initial automaton $\mathscr{A}^0$ contains only the states s^* and $s_\bot$ with transitions as described above. That is $\mathscr{A}^0 = (\{s^*, s_\bot\}, \Gamma, \delta, \{s_\bot\})$ with

$$\delta = \left\{ s^* \xrightarrow{A} \{s^*\} \mid A \in \Gamma \wedge A \neq \bot \right\} \cup \left\{ s^* \xrightarrow{\bot} \{s_\bot\} \right\} .$$

The algorithm is then a call to the function Fix$_0$($\mathscr{A}^0$) defined below. We define two functions for computing the fixed points for Z_0 and Z_1. Both of these functions are similar to each other: they begin by setting up an automaton representing the initial approximation of the fixed point, either by adding no transitions (the empty set) or all transitions (the largest set). They then enter a loop of computing the next approximation and then using projection to transfer (and accelerate) the new value to the states p^0 or p^1 as appropriate. The function Fix$_0$($\mathscr{A}$) computes the fixed point for Z_0 and uses Fix$_1$($\mathscr{A}$) to compute the next approximation, while Fix$_1$($\mathscr{A}$) computes the fixed point for Z_1 and uses a function Pre($\mathscr{A}$) to compute the next approximation. These two functions are thus defined

```
function Fix$_0$($\mathscr{A}$)
    $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
    $\mathbb{S}' \leftarrow \mathbb{S} \cup \{p^0 \mid p \in Q\}$
    $\delta' \leftarrow \left\{ p^0 \xrightarrow{A} S \mid p \in Q \wedge A \in \Gamma \wedge A \neq \bot \wedge S \subseteq \mathbb{S}' \setminus \{s_\bot\} \right\} \cup \left\{ p^0 \xrightarrow{\bot} \{s_\bot\} \mid p \in Q \right\}$
    $\mathscr{B} \leftarrow (\mathbb{S}', \Gamma, \delta', \mathscr{F})$
    repeat
        $\mathscr{B} \leftarrow$ Fix$_1$($\mathscr{B}$)
        $\mathscr{B} \leftarrow$ Proj($\mathscr{B}$, 1, 0)
    until $\mathscr{B}$ unchanged
    return $\mathscr{B}$
end function
```

and

```
function Fix$_1$($\mathscr{A}$)
    $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
    $\mathbb{S}' \leftarrow \mathbb{S} \cup \{p^1 \mid p \in Q\}$
    $\mathscr{B} \leftarrow (\mathbb{S}', \Gamma, \delta, \mathscr{F})$
    repeat
        $\mathscr{B} \leftarrow$ Pre($\mathscr{B}$)
        $\mathscr{B} \leftarrow$ Proj($\mathscr{B}'$, 2, 1)
    until $\mathscr{B}$ unchanged
```

```
        **return** $\mathscr{B}$
    **end function** .
```

The inner fixed point computation uses a function $\textsc{Pre}(\mathscr{A})$ to compute the step of the calculation corresponding to

$$(F \cap Pre_P(Z_0)) \cup \left(\overline{F} \cap Pre_P(Z_1)\right) .$$

This function adds transitions in the same way as the loop of saturation algorithm for reachability games, except it is sensitive to the two different fixed point variables. For convenience, we define the function Ω such that

$$\Omega(p) = \begin{cases} 0 & p \in F \\ 1 & p \notin F . \end{cases}$$

We can then define

```
    function $\textsc{Pre}(\mathscr{A})$
        $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
        $\mathbb{S}' \leftarrow \mathbb{S} \cup \{p^2 \mid p \in Q\}$
        $\delta' \leftarrow \begin{array}{l} \left\{p^2 \xrightarrow{A} S \;\middle|\; p \in Q_E \wedge \exists (p,a) \to (q,w) \in \Delta . q^{\Omega(p)} \xrightarrow{w} S\right\} \cup \\ \left\{p^2 \xrightarrow{A} S \;\middle|\; p \in Q_A \wedge \forall (p,a) \to (q,w) \in \Delta . \exists q^{\Omega(p)} \xrightarrow{w} S' . S' \subseteq S\right\} \end{array}$
        return $(\mathbb{S}', \Gamma, \delta', \mathscr{F})$
    end function .
```

The automaton $\mathscr{B}$ that is the result of $\textsc{Fix}_0(\mathscr{A}^0)$ will be such that $(p,w) \in \mathscr{W}$ iff $w \in \mathscr{L}_{p^0}(\mathscr{B})$. Since there are at most an exponential number of transitions in the automaton each fixed point may iterate at most an exponential number of times. This gives us an overall exponential run time for the algorithm.

Theorem 4 *The winning region of a pushdown Büchi game is regular and computable in exponential time.*

Note that for the one player case (*i.e.* all states belong to Éloise), the computation can be done in polynomial time [7, 23].

4.3.2 Winning strategies

Cachat also showed that, like in reachability games, it is possible to construct a linear space positional strategy and a constant time (though not positional) pushdown strategy for Éloise. However, in his PhD. thesis [14] Cachat observes that adopting his techniques for computing strategies for Abelard is not clear. However, it is known that, even for the full case of parity games, a pushdown strategy exists using different techniques [59, 42].

4.4 Pushdown parity games

Parity games allow more complex liveness properties to be checked. To define a parity game, each configuration is assigned a "colour" from a set of colours represented by natural numbers. The winner of the game depends on the smallest colour appearing infinitely often in the run: if it is even then Éloise wins the game, else Abelard wins.

More formally, given a sequence of configurations $\rho = (q_0, w_0), (q_1, w_1), \ldots$ let $\mathrm{Inf}(\rho)$ be the set of control states appearing infinitely often in ρ. That is

$$\mathrm{Inf}(\rho) = \left\{ q \;\middle|\; \forall i \exists j > i . q_j = q \right\} .$$

Given a set of control states Q and maximum colour κ, let $\Omega : Q \to \{0, \dots, \kappa\}$ be a colouring function assigning colours to each control state. We can generalise Ω to sets of control states P by taking the image of P. That is, $\Omega(P) = \{\alpha \mid \exists p \in P. \Omega(p) = \alpha\}$.

A pushdown parity game is a tuple $(Q, \Gamma, \perp, \Delta, \Omega)$ where $\Omega : Q \to \{0, \dots, \kappa\}$ is a colouring function assigning to each control state a colour from the set $\{0, \dots, \kappa\}$. Moreover, the tuple defines a pushdown game $(Q, \Gamma, \perp, \Delta, W)$ where

$$W = \{\rho \mid \min(\Omega(\mathrm{Inf}(\rho))) \text{ is even}\} \ .$$

Thus, a Büchi game is a special case of a parity game, where the set of colours is $\{0, 1\}$ and

$$\Omega(p) = \begin{cases} 0 & p \in F \\ 1 & p \notin F \ . \end{cases}$$

4.4.1 Characterising the winning region

The characterisation of Éloise's winning region in terms of fixed points is a natural extension of the Büchi version. That is, assuming κ to be odd and writing C_α to denote $\{(p, w) \mid \Omega(p) = \alpha\}$, we need

$$\nu Z_0 . \mu Z_1 . \cdots . \nu Z_{\kappa-1} . \mu Z_\kappa . \bigcup_{0 \leq \alpha \leq \kappa} (C_\alpha \cap Pre_P(Z_\alpha)) \ .$$

This formula can be understood as a generalisation of the Büchi formula, where $F = C_0$ and $\overline{F} = C_1$. When the colour of a configuration is odd, then it is bound by a least fixed point. Hence, it must eventually exit this fixed point by visiting a configuration with a smaller colour (just like a configuration in $\overline{F}$ had to visit a configuration in F). When the colour is even, then it is bound by a greatest fixed point – hence a play can stay within this fixed point, never visiting a smaller colour, and satisfy the winning condition for Éloise.

4.4.2 Computing the winning region

Fix a pushdown parity game $P = (Q, \Gamma, \perp, \Delta, \Omega)$. Computing the winning region in a pushdown parity game is a direct extension of the algorithm presented for Büchi games. Since a Büchi game is simply a pushdown parity game with two colours, we generalise the nesting of the fixed point calls to an arbitrary number of colours. To this end we introduce a function DISPATCH($\mathscr{A}$, α) that manages the level of nesting, and performs a fixed point or a pre-step analysis as appropriate.

```
function DISPATCH($\mathscr{A}$, $\alpha$)
    if $\alpha = \kappa + 1$ then
        return PRE($\mathscr{A}$)
    else
        return FIX($\mathscr{A}$, $\alpha$)
    end if
end function
```

Using this function we can define a generic fixed point function based on the Büchi functions. This function performs the nested calculations and the projection as before. The initial transitions from the new states introduced by the function depend on the parity of α: when computing an even (greatest) fixed point, we add all transitions, and when computing an odd (least) fixed point, we add no transitions.

```
function FIX($\mathscr{A}$, $\alpha$)
```

```
    $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
    $\mathbb{S}' \leftarrow \{p^\alpha \mid p \in Q\}$
    if $\alpha$ is even then
        $\delta' \leftarrow \left\{p^\alpha \xrightarrow{A} S \mid p \in Q \wedge A \in \Gamma \wedge A \neq \perp \wedge S \subseteq \mathbb{S}' \setminus \{s_\perp\}\right\} \cup \left\{p^\alpha \xrightarrow{\perp} \{s_\perp\} \mid p \in Q\right\}$
    else
        $\delta' \leftarrow \emptyset$
    end if
    $\mathscr{B} \leftarrow (\mathbb{S} \cup \mathbb{S}', \Gamma, \delta \cup \delta', \mathscr{F})$
    repeat
        $\mathscr{B} \leftarrow$ DISPATCH($\mathscr{B}$, $\alpha + 1$)
        $\mathscr{B} \leftarrow$ PROJ($\mathscr{B}$, $\alpha + 1$, $\alpha$)
    until $\mathscr{B}$ unchanged
     return $\mathscr{B}$
end function
```

Finally, we redefine the PRE($\mathscr{A}$) function to add transitions to the correct initial states. Note, we were already using Ω to distinguish between different fixed point variables, hence this function is almost identical to the Büchi case.

```
function PRE($\mathscr{A}$)
    $(\mathbb{S}, \Gamma, \delta, \mathscr{F}) \leftarrow \mathscr{A}$
    $\mathbb{S}' \leftarrow \mathbb{S} \cup \left\{p^{\kappa+1} \mid p \in Q\right\}$
    $\delta' \leftarrow \begin{array}{l} \left\{p^{\kappa+1} \xrightarrow{A} S \;\middle|\; p \in Q_E \wedge \exists (p,a) \rightarrow (q,w) \in \Delta . q^{\Omega(p)} \xrightarrow{w} S\right\} \cup \\ \left\{p^{\kappa+1} \xrightarrow{A} S \;\middle|\; p \in Q_A \wedge \forall (p,a) \rightarrow (q,w) \in \Delta . \exists q^{\Omega(p)} \xrightarrow{w} S' . S' \subseteq S\right\} \end{array}$
     return $(\mathbb{S}', \Gamma, \delta', \mathscr{F})$
end function
```

Thus, to compute the winning region of a pushdown parity game, we make the call DISPATCH($\mathscr{A}^0$, 0) where $\mathscr{A}^0$ is the initial automaton with only the states s^* and $s_\perp$ as defined in the Büchi case.

The automaton $\mathscr{B}$ that is the result of DISPATCH($\mathscr{A}^0$, 0) will be such that $(p, w) \in \mathscr{W}$ iff $w \in \mathscr{L}_{p^0}(\mathscr{B})$. Since there are at most an exponential number of transitions in the automaton each fixed point may iterate at most an exponential number of times. This gives us an overall exponential run time for the algorithm.

Theorem 5 *The winning region of a pushdown parity game is regular and computable in exponential time.*

4.4.3 Winning strategies

Unfortunately, it is currently unknown how to compute the winning strategies for Éloise and Abelard using the saturation technique for pushdown parity games. However, using a different approach, both Walukiewicz [59] and Serre [42] have shown that a pushdown strategy exists for both players.

5 Implementations and Applications of Saturation Methods

In this article, we have presented the saturation method from a theoretical standpoint. The method, however, is an algorithmic approach that is well suited to implementation, and several tools have been constructed using saturation as its core technique.

5.1 Single Player Implementations

Perhaps the most famous of these tools is Moped [22, 40] and its incarnation as a model checker for Java, JMoped [54, 52]. In taking the algorithm from a theoretical tool to a practical one, a number of new concerns had to be taken into account.

The rules of a pushdown system roughly correspond to the statements in a program. In a program with thousands of lines, a fixed point iteration that checks, during each iteration, whether each rule leads to new transitions in the automaton would be woefully inefficient. In constructing Moped, Esparza *et al.* [21] showed how this naive outer loop can be reorganised such that, at each iteration, only the relevant rules of the system were considered, leading to a significant improvement in performance.

A second consideration of applications to the analysis of program models is the handling of data values. Boolean programs are essentially pushdown systems where each control state and stack character contains a valuation of a set of global and local boolean variables respectively. These boolean programs are the natural output of predicate abstraction tools such as SATABS [18] as well as the target compilation language of JMoped.

Since there are only finitely many valuations of sets of boolean variables, they can directly be encoded as control states or characters and standard pushdown analysis techniques can be employed. However, since they are also exponential in number, such an approach is inherently inefficient. Hence, Esparza *et al.* introduced *symbolic pushdown systems* [22] which make boolean valuations first class objects. The saturation technique was extended by adding BDDs representing variable valuations to the edges of the P-automata, leading to an implementation capable of analysing symbolic pushdown systems derived from real-world programs.

Around this time it was observed by Reps that the BDDs could be replaced by any abstract domain of values that was sufficiently well behaved, and many static analyses could be derived. This led to the introduction of *weighted pushdown systems* [38] (and, indeed, *extended* weighted pushdown systems amongst other improvements [34, 33]), of which symbolic pushdown systems and their BDD representation were an instance. The developers of Moped created the *weighted pushdown system library* [60] as a component of Moped, and Reps *et al.* developed WALi [58] implementing these new algorithms.

5.2 Two-Player Implementations

Perhaps the most straight-forward optimisation to make to the saturation technique as presented for two-player games is via the observation that a transition

$$s \xrightarrow{A} S$$

is effectively redundant if there exists another transition

$$s \xrightarrow{A} S'$$

with $S' \subseteq S$. This is because an accepting run from S contains within it an accepting run from S', and thus the former transition can be removed.

When considering reachability games, it is also possible to improve the naive fixed point iteration, as in the single-player case, to avoid checking against all pushdown rules during each step of the implementation. Such an optimisation was introduced by Suwimonteerabuth *et al.* and implemented with applications to certificate chain analysis [55].

This work has recently been built upon by Song who has developed various tools based upon reductions to Büchi games and tools for their analysis. Primarily this work has focussed on a specification language that is an extension of CTL and its translation into symbolic pushdown Büchi games [46, 48] resulting in the tool PuMoC [47]. The main application of this work has been in the detection of malware. More recently still, this work has been developed for LTL-like properties to deal with situations where the CTL approach was insufficient [49], culminating in the PoMMaDe tool [51].

However, the combination of BDD representations and alternating automata is not an easy one, since BDDs lack the necessary alternation for a direct embedding. Hence, Song's algorithm pays an extra exponential in its worst-case complexity (doubly exponential rather than exponential), although the practical runtime is improved. The optimal inclusion of symbolic representations into the analysis of two-player games remains an open problem.

The saturation technique for the full case of parity games has been implemented in the PDSolver tool [28] and applied to dataflow analysis problems for Java programs. Due to the interactions between the several layers of fixed points, it is not clear how to adapt Esparza *et al.* and Suwimonteerabuth *et al.*'s efficient algorithms to this case, nor how to include symbolic representations. These remain limitations of the tool, and interesting avenues for future work.

6 Extensions of the Saturation Method

In this article we have looked at the different saturation methods for pushdown systems. Across several articles, the technique has proved to be applicable to various extensions to the basic model. We briefly list some of these results here.

Concurrency The reachability problem for pushdown systems with two or more stacks is well known to be undecidable. Since multiple stacks are needed to model multi-thread recursive programs, a number of underapproximation techniques have been studied for which the reachability problem is decidable. One such technique is *bounded context switching* [37] where the number of interactions between the threads is limited to an *a priori* fixed number k. While this cannot prove the absence of errors, it is effective at finding bugs in programs, since, empirically, bugs usually manifest themselves within a small number of interactions. This restriction can be relaxed further by allowing a bounded number of *phases* [56] (where all threads run concurrently, but during each phase only one thread is allowed to pop from its stack), or a bounded scope [57] (where, threads are scheduled in a round-robin fashion, and characters may only be removed from the stack if they were pushed at most a fixed number of rounds earlier).

The saturation technique has proved useful for each of these restrictions. In particular, Moped has been extended to provide context bounded analysis of multi-stack pushdown systems [53] by Suwimonteerabuth *et al.* and saturation was used by Seth to provide a regular solution to the global reachability problem for phase bounded pushdown systems [45]. The original proof that the reachability problem for scope bounded pushdown systems is decidable was itself an extension of the saturation technique [57].

An alternative restriction that permits a decidable reachability and LTL model checking problem is that of *ordered multi-pushdown systems* where only the leftmost non-empty stack is able to remove characters. Atig provides two extensions of the saturation technique in this direction [1]. First, instead of each pushdown rule adding a fixed sequence of characters to the stack, he allows rules to contain languages of sequences that may be pushed. If it is decidable whether the language of a rule intersected with a regular language is empty, then an augmented saturation technique leads to an effective analysis algorithm. In particular, the model checking problem for ordered pushdown systems can be solved with this formalism.

Finally, Song generalises his LTL model checking algorithms to the case of pushdown systems with dynamic thread creation [50], again using a saturation technique at its core.

Ground Tree Rewrite Systems and Resources Ground tree rewrite systems can be thought of as pushdown systems with a single control state and a more complex stack structure. That is, the stack is a tree rather than a word. Rewrite rules in this system replace complete subtrees. For example a push rule $(p,A) \to (p,BC)$ can be considered to be replacing the subtree consisting in the leaf node A with the subtree $B(C)$ (i.e. a B-node with a C-leaf as a child). In 1987, Dauchet *et al.* used saturation to show that the confluence problem for these systems is decidable [19]. More recently, Lang and Löding adapted this method to analyse prefix replacement systems with resource usage [35].

Higher-Order and Collapsible Pushdown Systems Pushdown systems provide a natural model for first-order recursive programs. When considering higher-order programs, we can use *higher-order pushdown systems* [36] whose stacks have a nested "stack-of-stacks" structure. These systems correspond to *higher-order recursion schemes* satisfying a *safety* constraint [31]. Recently, these systems were generalised to *collapsible pushdown systems* (via *panic automata* [32]), providing an automata model without the need for the safety constraint [26].

The saturation technique was first applied to the analysis of higher-order systems by Bouajjani and Meyer [8] who considered higher-order pushdown systems with a single control state. This algorithm was generalised by Hague and Ong to permit an arbitrary number of control states [27]. An alternative construction in the case of second order higher-order pushdown systems was provided by Seth [43].

More recently this approach was developed by Broadbent *et al.* to obtain a saturation algorithm for the full case of collapsible pushdown systems [9], leading to the analysis tool C-SHORe [10]. This algorithm was applied directly to the analysis of recursion schemes (without the intermediate automata model) by Broadbent and Kobayashi, resulting in the HorSat tool [11].

Finally, the case of concurrent higher-order systems has been briefly considered. Seth used saturation to show that parity games over phase-bounded higher-order pushdown systems (without collapse) are effectively solvable [44]. Recently, Hague showed that the saturation approaches for first-order phase-bounded, ordered and scope-bounded pushdown systems can be adapted to solve the analogous reachability problems for collapsible pushdown systems [25].

References

[1] M. F. Atig (2012): *Model-Checking of Ordered Multi-Pushdown Automata. Logical Methods in Computer Science* 8(3), doi:10.2168/LMCS-8(3:20)2012. Available at `http://dx.doi.org/10.2168/LMCS-8(3:20)2012`.

[2] T. Ball, V. Levin & S. K. Rajamani (2011): *A decade of software model checking with SLAM. Commun. ACM* 54(7), pp. 68–76, doi:10.1145/1965724.1965743. Available at `http://doi.acm.org/10.1145/1965724.1965743`.

[3] T. Ball & S. K. Rajamani (2000): *Bebop: A Symbolic Model Checker for Boolean Programs*. In: *SPIN*, pp. 113–130, doi:10.1007/10722468_7. Available at `http://dx.doi.org/10.1007/10722468_7`.

[4] Y. Bar-Hillel, M. Perles & E. Shamir (1961): *On formal properties of simple phrase structure grammars. Z. Phonetik Sprachwiss. Kommunikat.* 14, p. 143172.

[5] M. Benois (1969): *Parties rationnelles du groupe libre. Comptes-Rendus de l'Acamdémie des Sciences de Paris, Série A* 269, pp. 1188–1190.

[6] M. Benois & J. Sakarovitch (1986): *On the Complexity of Some Extended Word Problems Defined by Cancellation Rules*. *Inf. Process. Lett.* 6, pp. 281–287, doi:10.1016/0020-0190(86)90087-6. Available at `http://dx.doi.org/10.1016/0020-0190(86)90087-6`.

[7] A. Bouajjani, J. Esparza & O. Maler (1997): *Reachability Analysis of Pushdown Automata: Application to Model-Checking*. In: *CONCUR*, pp. 135–150, doi:10.1007/3-540-63141-0_10. Available at `http://dx.doi.org/10.1007/3-540-63141-0_10`.

[8] A. Bouajjani & A. Meyer (2004): *Symbolic Reachability Analysis of Higher-Order Context-Free Processes*. In: *FSTTCS*, pp. 135–147, doi:10.1007/978-3-540-30538-5_12. Available at `http://dx.doi.org/10.1007/978-3-540-30538-5_12`.

[9] C. H. Broadbent, A. Carayol, M. Hague & O. Serre (2012): *A Saturation Method for Collapsible Pushdown Systems*. In: *ICALP*, pp. 165–176, doi:10.1007/978-3-642-31585-5_18. Available at `http://dx.doi.org/10.1007/978-3-642-31585-5_18`.

[10] C. H. Broadbent, A. Carayol, M. Hague & O. Serre (2013): *C-SHORe: a collapsible approach to higher-order verification*. In: *ICFP*, pp. 13–24, doi:10.1145/2500365.2500589. Available at `http://doi.acm.org/10.1145/2500365.2500589`.

[11] C. H. Broadbent & N. Kobayashi (2013): *Saturation-Based Model Checking of Higher-Order Recursion Schemes*. In: *CSL*, pp. 129–148, doi:10.4230/LIPIcs.CSL.2013.129. Available at `http://dx.doi.org/10.4230/LIPIcs.CSL.2013.129`.

[12] R. J Büchi (1964): *Regular canonical systems*. *Archive for Mathematical Logic* 6(3), pp. 91–111, doi:10.1007/BF01969548.

[13] T. Cachat (2002): *Symbolic Strategy Synthesis for Games on Pushdown Graphs*. In: *ICALP*, pp. 704–715, doi:10.1007/3-540-45465-9_60. Available at `http://dx.doi.org/10.1007/3-540-45465-9_60`.

[14] T. Cachat (2003): *Games on Pushdown Graphs and Extensions*. Ph.D. thesis, RWTH Aachen. Available at `http://www.liafa.jussieu.fr/~txc/Download/Cachat-PhD.pdf`.

[15] D. Caucal (1988): *Récritures suffixes de mots*. Research Report RR-0871, INRIA.

[16] D. Caucal (1990): *On the Regular Structure of Prefix Rewriting*. In: *Proceedings of the 15th Colloquium on Trees in Algebra and Programming (CAAP'90)*, *Lecture Notes in Computer Science* 431, Springer, pp. 87–102, doi:10.1007/3-540-52590-4_42.

[17] D. Caucal (2008): *Deterministic graph grammars*. In Jörg Flum, Erich Grädel & Thomas Wilke, editors: *Logic and Automata: History and Perspectives in Honor of Wolfgang Thomas*, *Texts in Logic and Games* 2, Amsterdam University Press, pp. 169–250.

[18] E. M. Clarke, D. Kroening, N. Sharygina & K. Yorav (2005): *SATABS: SAT-Based Predicate Abstraction for ANSI-C*. In: *TACAS*, pp. 570–574. Available at `http://dx.doi.org/10.1007/978-3-540-31980-1_40`.

[19] M. Dauchet, S. Tison, T. Heuillard & P. Lescanne (1987): *Decidability of the Confluence of Ground Term Rewriting Systems*. In: *LICS*, pp. 353–359.

[20] J. Esparza, D. Hansel, P. Rossmanith & S. Schwoon (2000): *Efficient Algorithm for Model Checking Pushdown Systems*. In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, *LNCS* 1885, Springer, pp. 232–247, doi:10.1007/1072216720.

[21] J. Esparza, D. Hansel, P. Rossmanith & S. Schwoon (2000): *Efficient Algorithms for Model Checking Pushdown Systems*. In: *CAV*, pp. 232–247, doi:10.1007/10722167_20. Available at `http://dx.doi.org/10.1007/10722167_20`.

[22] J. Esparza & S. Schwoon (2001): *A BDD-Based Model Checker for Recursive Programs*. In: *CAV*, pp. 324–336, doi:10.1007/3-540-44585-4_30. Available at `http://dx.doi.org/10.1007/3-540-44585-4_30`.

[23] A. Finkel, B. Willems & P. Wolper (1997): *A direct symbolic approach to model checking pushdown systems*. *Electr. Notes Theor. Comput. Sci.* 9, pp. 27–37, doi:10.1007/3-540-45465-9_60. Available at `http://dx.doi.org/10.1016/S1571-0661(05)80426-8`.

[24] S. A. Greibach (1967): *A note on pushdown store automata and regular systems*. Proceedings of the American Mathematical Society, pp. 263–268, doi:10.1090/S0002-9939-1967-0209086-1.

[25] M. Hague (2013): *Saturation of Concurrent Collapsible Pushdown Systems*. In: FSTTCS, pp. 313–325, doi:10.4230/LIPIcs.FSTTCS.2013.313. Available at `http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2013.313`.

[26] M. Hague, A. S. Murawski, C.-H. Luke Ong & O. Serre (2008): *Collapsible Pushdown Automata and Recursion Schemes*. In: LICS, pp. 452–461, doi:10.1109/LICS.2008.34. Available at `http://doi.ieeecomputersociety.org/10.1109/LICS.2008.34`.

[27] M. Hague & C.-H. L. Ong (2008): *Symbolic Backwards-Reachability Analysis for Higher-Order Pushdown Systems*. Logical Methods in Computer Science 4(4), doi:10.2168/LMCS-4(4:14)2008. Available at `http://dx.doi.org/10.2168/LMCS-4(4:14)2008`.

[28] M. Hague & C.-H. L. Ong (2010): *Analysing Mu-Calculus Properties of Pushdown Systems*. In: SPIN, pp. 187–192, doi:10.1007/978-3-642-16164-3_14. Available at `http://dx.doi.org/10.1007/978-3-642-16164-3_14`.

[29] M. Hague & C.-H. Luke Ong (2009): *Winning Regions of Pushdown Parity Games: A Saturation Method*. In: CONCUR, pp. 384–398, doi:10.1007/978-3-642-04081-8_26. Available at `http://dx.doi.org/10.1007/978-3-642-04081-8_26`.

[30] N. D. Jones & S. S. Muchnick (1977): *Even Simple Programs Are Hard To Analyze*. J. ACM 24(2), pp. 338–350, doi:10.1145/322003.322016. Available at `http://doi.acm.org/10.1145/322003.322016`.

[31] T. Knapik, D. Niwinski & P. Urzyczyn (2002): *Higher-Order Pushdown Trees Are Easy*. In: FoSSaCS, pp. 205–222, doi:10.1007/3-540-45931-6_15. Available at `http://dx.doi.org/10.1007/3-540-45931-6_15`.

[32] T. Knapik, D. Niwinski, P. Urzyczyn & I. Walukiewicz (2005): *Unsafe Grammars and Panic Automata*. In: ICALP, pp. 1450–1461, doi:10.1007/11523468_117. Available at `http://dx.doi.org/10.1007/11523468_117`.

[33] A. Lal & T. W. Reps (2006): *Improving Pushdown System Model Checking*. In: CAV, pp. 343–357, doi:10.1007/11817963_32. Available at `http://dx.doi.org/10.1007/11817963_32`.

[34] A. Lal, T. W. Reps & G. Balakrishnan (2005): *Extended Weighted Pushdown Systems*. In: CAV, pp. 434–448, doi:10.1007/11513988_44. Available at `http://dx.doi.org/10.1007/11513988_44`.

[35] M. Lang & C. Löding (2013): *Modeling and Verification of Infinite Systems with Resources*. Logical Methods in Computer Science 9(4), doi:10.2168/LMCS-9(4:22)2013. Available at `http://dx.doi.org/10.2168/LMCS-9(4:22)2013,http://arxiv.org/abs/1311.1043`.

[36] A. N. Maslov (1976): *Multilevel stack automata*. Problems of Information Transmission 15, pp. 1170–1174.

[37] S. Qadeer (2008): *The Case for Context-Bounded Verification of Concurrent Programs*. In: Proceedings of the 15th international workshop on Model Checking Software, SPIN '08, Springer-Verlag, Berlin, Heidelberg, pp. 3–6, doi:10.1007/978-3-540-85114-1_2. Available at `http://dx.doi.org/10.1007/978-3-540-85114-1_2`.

[38] T. W. Reps, S. Schwoon, S. Jha & D. Melski (2005): *Weighted pushdown systems and their application to interprocedural dataflow analysis*. Sci. Comput. Program. 58(1-2), pp. 206–263. Available at `http://dx.doi.org/10.1016/j.scico.2005.02.009`.

[39] J. Sakarovitch (2009): *Elements of Automata Theory*. Cambridge University Press.

[40] S. Schwoon (2002): *Model-checking Pushdown Systems*. Ph.D. thesis, Technical University of Munich.

[41] S. Schwoon (2002): *Model-Checking Pushdown Systems*. Ph.D. thesis, Technische Universität München.

[42] O. Serre (2004): *Contribution à létude des jeux sur des graphes de processus à pile*. Ph.D. thesis, Université Paris 7 – Denis Diderot, UFR dinformatique. Available at `http://tel.archives-ouvertes.fr/tel-00011326`.

[43] A. Seth (2008): *An Alternative Construction in Symbolic Reachability Analysis of Second Order Pushdown Systems*. *Int. J. Found. Comput. Sci.* 19(4), pp. 983–998, doi:10.1142/S012905410800608X. Available at `http://dx.doi.org/10.1142/S012905410800608X`.

[44] A. Seth (2009): *Games on Higher Order Multi-stack Pushdown Systems*. In: *RP*, pp. 203–216, doi:10.1007/978-3-642-04420-5_19. Available at `http://dx.doi.org/10.1007/978-3-642-04420-5_19`.

[45] A. Seth (2010): *Global Reachability in Bounded Phase Multi-stack Pushdown Systems*. In: *CAV*, pp. 615–628, doi:10.1007/978-3-642-14295-6_53. Available at `http://dx.doi.org/10.1007/978-3-642-14295-6_53`.

[46] F. Song & T. Touili (2011): *Efficient CTL Model-Checking for Pushdown Systems*. In: *CONCUR*, pp. 434–449, doi:10.1007/978-3-642-23217-6_29. Available at `http://dx.doi.org/10.1007/978-3-642-23217-6_29`.

[47] F. Song & T. Touili (2012): *PuMoC: a CTL model-checker for sequential programs*. In: *ASE*, pp. 346–349, doi:10.1145/2351676.2351743. Available at `http://doi.acm.org/10.1145/2351676.2351743`.

[48] F. Song & T. Touili (2012): *Pushdown Model Checking for Malware Detection*. In: *TACAS*, pp. 110–125, doi:10.1007/978-3-642-28756-5_9. Available at `http://dx.doi.org/10.1007/978-3-642-28756-5_9`.

[49] F. Song & T. Touili (2013): *LTL Model-Checking for Malware Detection*. In: *TACAS*, pp. 416–431, doi:10.1007/978-3-642-36742-7_29. Available at `http://dx.doi.org/10.1007/978-3-642-36742-7_29`.

[50] F. Song & T. Touili (2013): *Model Checking Dynamic Pushdown Networks*. In: *APLAS*, pp. 33–49, doi:10.1007/978-3-319-03542-0_3. Available at `http://dx.doi.org/10.1007/978-3-319-03542-0_3`.

[51] F. Song & T. Touili (2013): *PoMMaDe: pushdown model-checking for malware detection*. In: *ESEC/SIGSOFT FSE*, pp. 607–610, doi:10.1145/2491411.2494599. Available at `http://doi.acm.org/10.1145/2491411.2494599`.

[52] D. Suwimonteerabuth, F. Berger, S. Schwoon & J. Esparza (2007): *jMoped: A Test Environment for Java Programs*. In: *CAV*, pp. 164–167, doi:10.1007/978-3-540-73368-3_19. Available at `http://dx.doi.org/10.1007/978-3-540-73368-3_19`.

[53] D. Suwimonteerabuth, J. Esparza & S. Schwoon (2008): *Symbolic Context-Bounded Analysis of Multithreaded Java Programs*. In: *SPIN*, pp. 270–287, doi:10.1007/978-3-540-85114-1_19. Available at `http://dx.doi.org/10.1007/978-3-540-85114-1_19`.

[54] D. Suwimonteerabuth, S. Schwoon & J. Esparza (2005): *jMoped: A Java Bytecode Checker Based on Moped*. In: *TACAS*, pp. 541–545, doi:10.1007/978-3-540-31980-1_35. Available at `http://dx.doi.org/10.1007/978-3-540-31980-1_35`.

[55] D. Suwimonteerabuth, S. Schwoon & J. Esparza (2006): *Efficient Algorithms for Alternating Pushdown Systems with an Application to the Computation of Certificate Chains*. In: *ATVA*, pp. 141–153, doi:10.1007/11901914_13. Available at `http://dx.doi.org/10.1007/11901914_13`.

[56] S. La Torre, P. Madhusudan & G. Parlato (2007): *A Robust Class of Context-Sensitive Languages*. In: *LICS*, pp. 161–170, doi:10.1109/LICS.2007.9. Available at `http://doi.ieeecomputersociety.org/10.1109/LICS.2007.9`.

[57] S. La Torre & M. Napoli (2011): *Reachability of Multistack Pushdown Systems with Scope-Bounded Matching Relations*. In: *CONCUR*, pp. 203–218, doi:10.1007/978-3-642-23217-6_14. Available at `http://dx.doi.org/10.1007/978-3-642-23217-6_14`.

[58] WALi: Weighted Automata Library: `https://research.cs.wisc.edu/wpis/wpds/download.php`.

[59] I. Walukiewicz (2001): *Pushdown Processes: Games and Model-Checking*. *Inf. Comput.* 164(2), pp. 234–263, doi:10.1006/inco.2000.2894. Available at `http://dx.doi.org/10.1006/inco.2000.2894`.

[60] WPDS Library: `http://www2.informatik.uni-stuttgart.de/fmi/szs/tools/wpds/`.