Complete SAT-based Model Checking for Context-Free Processes*

Geng-Dian Huang[1,2] and Bow-Yaw Wang[1]

[1] Institute of Information Science
Academia Sinica, Taiwan
[2] Department of Electrical Engineering
National Taiwan University, Taiwan
[gdhuang|bywang]@iis.sinica.edu.tw

Abstract. A complete SAT-based model checking algorithm for context-free processes is presented. We reduce proof search in local model checking to Boolean satisfiability. Bounded proof search can therefore be performed by SAT solvers. Moreover, the completion of proof search is reduced to Boolean unsatisfiability and hence can be checked by SAT solvers. By encoding the local model checking algorithm in [13], SAT solvers are able to verify properties in the universal fragment of alternation-free μ-calculus formula on context-free processes.

1 Introduction

Since pushdown systems give natural representations of program control flows, problems in program analysis can be reduced to verification problems on the infinite-state model. In the past years, efficient verification algorithms for pushdown systems have been proposed [11, 12]. Experimental results suggest that the BDD-based algorithm for the succinct model could be much more space-efficient than those for finite-state systems in program analysis [12].

Meanwhile, hardware verification has been influenced by the development of practical satisfiability (SAT) solvers [4, 3]. Thanks to various heuristics, SAT solvers are very efficient in both time and space. By reducing bounded verification problems to Boolean satisfiability, the technique can detect flaws in finite-state models unattainable by explicit-state or BDD-based algorithms.

SAT-based verification algorithms for finite-state systems make the experiment in [12] regretfully obsolete. Since `bebop` uses a BDD-based algorithm [2], it is unclear how the explicit-state [5, 11] or BDD-based [12] algorithms for pushdown systems compare with SAT-based algorithms for finite-state systems. Moreover, the explicit-state and BDD-based algorithms for pushdown systems might suffer from the same capacity problem as in finite-state systems. An SAT-based algorithm for pushdown systems could be more scalable.

* The work is partly supported by NSC grands 95-3114-P-001-002-Y02, 95-2221-E-001-024-MY3, and the SISARL thematic project of Academia Sinica.

In this paper, we give a complete SAT-based model checking algorithm for the universal fragment of alternation-free μ-calculus formulae on context-free processes. Given a context-free grammar, one may view derivations as system evolutions. A context-free grammar thus defines the transition system of a context-free process [6, 13]. Although the languages recognized by context-free grammars and pushdown automata coincide, pushdown systems are in fact more expressive than context-free processes [8]. Nevertheless, pushdown systems with only one control state are context-free processes. Problems in program analysis can thus be modeled by context-free processes. Moreover, our preliminary experimental results show that the new algorithm performs better than a BDD-based algorithm for large random models. We feel that our SAT-based verification algorithm could still be useful in program analysis.

Based on the explicit-state algorithm in [6], a local model checking algorithm for alternation-free μ-calculus formulae on context-free processes is developed [13]. We construct a Boolean formula whose satisfiability is equivalent to a bounded proof. If no proof within certain bounds can be found, the completion of proof search is then established by the unsatisfiability of another formula. Our SAT-based model checking algorithm therefore reduces proof search of the local model checking algorithm in [13] to Boolean (un)satisfiability. The universal fragment of alternation-free μ-calculus formulae on context-free processes can hence be verified by the absence of proofs of their negations using SAT solvers.

An explicit-state model checking algorithm for context-free processes is given in [6]. Second-order assertions specify properties on sets of states under contextual assumptions. Since the given property is of main concern, formulae in its closure are sufficient for contextual assumptions. The model checking problem is solved by computing contextual assumptions on finite representations. Employing second-order assertions, a local model checking algorithm for alternation-free μ-calculus formulae on context-free processes is developed in [13].

Complete SAT-based model checking algorithms for finite-state models can be found in literature [15, 1, 18]. Interpolation is exploited to verify invariants [15]. In [1], SAT solvers are used to detect cycles and check properties in linear temporal logic. A similar technique based on local model checking is able to verify the universal fragment of μ-calculus properties by SAT solvers [18].

Verification algorithms for pushdown systems have also been proposed [16, 5, 11, 12, 17]. Model checking monadic second-order logic properties is known to be decidable but with a non-elementary upper bound [16]. Verifying μ-calculus properties is DEXPTIME-complete for pushdown systems [5]. For linear properties, the problem can be solved in polynomial time but requires polynomial space [11]. A BDD-based algorithm is compared with the software verification tool `bebop` in [12]. Finally, a game-theoretic algorithm is given in [17].

The paper is organized as follows. Section 2 gives backgrounds. Our reduction of proof search to Boolean satisfiability is presented in Section 3. The SAT-based model checking algorithm for the universal fragment of alternation-free μ-calculus formulae is shown in Section 4. Preliminary experimental results are reported in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

A context-free process is a finite-state automaton with procedure calls and two designated locations.[3] Procedure invocation is denoted by names. The unique entry and exit points of procedures are represented by the start and end locations respectively.

Definition 1. *A* context-free process $P = \langle \Sigma, N, Act, \rightarrow_P, \delta, \epsilon \rangle$ *is a tuple where*

- Σ *is a finite set of* locations*;*
- N *and* Act *are finite sets of* names *and* actions *respectively;*
- $\rightarrow_P \subseteq \Sigma \times (N \cup Act) \times \Sigma$ *is its* transition relation*; and*
- δ *and* ϵ *are are the* start *and* end *locations respectively.*

For clarity, we write $\sigma \xrightarrow{\alpha} \sigma'$ for $(\sigma, \alpha, \sigma') \in \rightarrow$. A context-free process is *guarded* if for all $\delta \xrightarrow{\alpha} \sigma$, we have $\alpha \in Act$. We only consider guarded context-free processes in the following presentation.

A context-free process system is a set of recursively defined context-free processes. Let $\underline{n}$ be the name set $\{0, 1, \ldots, n\}$ and the name i denote the invocation of process P_i. Since all context-free processes share the same name set, mutual recursion can be modeled easily. In our setting, context-free processes and basic process algebra are in fact equivalent [9, 7].

Definition 2. *A* context-free process system $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ *consists of context-free processes* $P_0, \ldots, P_n$ *where*

- $P_0, \ldots, P_n$ *share the sets of names* $\underline{n}$ *and actions* Act*;*
- P_0 *is the* main process.

A context-free process system serves as a finite representation of a process graph. A process graph is a transition system with designated start and end states, and may have an infinite number of states.

Definition 3. *A* process graph $G = \langle S, Act, \rightarrow, s_0, s_e \rangle$ *is a tuple where*

- S *is the set of* states*;*
- Act *is the finite set of* actions*;*
- $\rightarrow \subseteq S \times Act \times S$ *is its* transition relation*; and*
- s_0 *and* s_e *are the* start *and* end *states respectively.*

The process graph represented by a context-free system is obtained by expanding recursive calls. Observe that copies of context-free processes can be made infinitely many times. A location in a context-free process may correspond to an infinite number of states in the process graph.

Definition 4. *Let* $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ *be a context-free process system with* $P_i = \langle \Sigma_i, \underline{n}, Act, \rightarrow_i, \delta_i, \epsilon_i \rangle$ *for* $0 \leq i \leq n$*. The* process graph $PG(\mathbb{P})$ of $\mathbb{P}$ *is obtained by expanding* $s_0 \xrightarrow{0} s_e$ *recursively as follows.*

[3] The term "location" is called "state class" in [6, 13].

1. *For each transition* $s \xrightarrow{i} s'$*, make a copy of the context-free process* P_i*; and*
2. *Identify* δ_i *and* ϵ_i *with* s *and* s' *respectively.*

When a copy of P_i is made, a state s is added for each location σ in Σ_i except δ_i and ϵ_i. We hence say s is an *instance* of σ. The notation $s \in \sigma$ denotes that s is an instance of σ or s is identified with σ when $\sigma = \epsilon_i$. Further, s'' is the *return state* of s (denoted by $end(s)$) if s is added while expanding $s' \xrightarrow{p_i} s''$. Both $end(s_0)$ and $end(s_e)$ are defined to be s_e.

Given the set Var of *relational variables*, $X \in Var$, and $A \subseteq Act$. The syntax of μ-calculus formulae is defined as follows

$$\phi ::= \mathsf{tt} \big| X \big| \neg\phi \big| \phi_0 \wedge \phi_1 \big| \langle A \rangle \phi \big| \mu X.\phi$$

Relational variables must be bound positively by least fixed point operators in $\mu X.\phi$. We adopt the following abbreviation: ff for $\neg\mathsf{tt}$, $\phi_0 \vee \phi_1$ for $\neg(\neg\phi_0 \wedge \neg\phi_1)$, $[A]\phi$ for $\neg\langle A\rangle\neg\phi$, and $\nu X.\phi$ for $\neg\mu X.\neg\phi[\neg X/X]$. A μ-calculus formula is *alternation-free* if all its fixed point subformulae do not have free relational variables bound by fixed point operators of the other type. The *negative normal form* of a μ-calculus formula is obtained by applying De Morgan's laws repeatedly so that negations appear only before tt. The *universal fragment of* μ*-calculus formulae* consists of μ-calculus formulae whose negative normal forms do not have existential modal operators ($\langle a\rangle\bullet$). Let ψ be a universal μ-calculus formula. It is easy to verify that the negative normal form of $\neg\psi$ does not have universal modal operators ($[a]\bullet$). In the following, we assume all formulae are in their negative normal forms.

Given a process graph $G = \langle S, Act, \rightarrow, s_0, s_e\rangle$, an environment e is a mapping from Var to 2^S. The notation $e[X \mapsto U]$ denotes the environment that maps X to U and Y to $e(Y)$ for $Y \neq X$. The semantic function $[\![\phi]\!]_e^G$ for the μ-calculus formula ϕ is defined as follows.

$$\begin{aligned}
[\![\mathsf{tt}]\!]_e^G &= S \\
[\![X]\!]_e^G &= e(X) \\
[\![\phi_0 \wedge \phi_1]\!]_e^G &= [\![\phi_0]\!]_e^G \cap [\![\phi_1]\!]_e^G \\
[\![\phi_0 \vee \phi_1]\!]_e^G &= [\![\phi_0]\!]_e^G \cup [\![\phi_1]\!]_e^G \\
[\![[A]\phi]\!]_e^G &= \{s \in S \mid \forall a, s'.a \in A \wedge s \xrightarrow{a} s' \Longrightarrow s' \in [\![\phi]\!]_e^G\} \\
[\![\langle A\rangle\phi]\!]_e^G &= \{s \in S \mid \exists a, s'.a \in A \wedge s \xrightarrow{a} s' \wedge s' \in [\![\phi]\!]_e^G\} \\
[\![\nu X.\phi]\!]_e^G &= \bigcup\{U \subseteq S \mid U \subseteq [\![\phi]\!]_{e[X\mapsto U]}^G\} \\
[\![\mu X.\phi]\!]_e^G &= \bigcap\{U \subseteq S \mid U \supseteq [\![\phi]\!]_{e[X\mapsto U]}^G\}
\end{aligned}$$

We say s *satisfies* ϕ *in process graph* G (denoted by $G, s \models \phi$) if $s \in [\![\phi]\!]_\emptyset^G$. Let Φ be a set of μ-calculus formulae. Define $G, s \models \Phi$ if $G, s \models \phi$ for all $\phi \in \Phi$. If $\mathbb{P}$ is a context-free system, we say $\mathbb{P}$ *satisfies* ϕ, $\mathbb{P} \models \phi$, if $PG(\mathbb{P}), s_0 \models \phi$. When there is no ambiguity, we write $s \models \phi$ and $s \models \Phi$ for $G, s \models \phi$ and $G, s \models \Phi$.

Since different instances of a location may be instantiated in different invocations, one cannot naively expect all instances to satisfy the same property. Contextual assumptions are hence used in the specification of locations. They are chosen from closures of μ-calculus formulae and postulated during process invocation [6, 13].

Definition 5. *The* closure $CL(\phi)$ *of a μ-calculus formula ϕ is inductively defined as follows.*

$$\begin{aligned}
CL(\mathtt{tt}) &= \emptyset \\
CL(\phi_0 \wedge \phi_1) &= \{\phi_0 \wedge \phi_1\} \cup CL(\phi_0) \cup CL(\phi_1) \\
CL(\phi_0 \vee \phi_1) &= \{\phi_0 \vee \phi_1\} \cup CL(\phi_0) \cup CL(\phi_1) \\
CL([A]\phi) &= \{[A]\phi\} \cup CL(\phi) \\
CL(\langle A\rangle\phi) &= \{\langle A\rangle\phi\} \cup CL(\phi) \\
CL(\nu X.\phi) &= \{\nu X.\phi\} \cup CL(\phi[\nu X.\phi/X]) \\
CL(\mu X.\phi) &= \{\mu X.\phi\} \cup CL(\phi[\mu X.\phi/X])
\end{aligned}$$

Given a μ-calculus formula ϕ and a set of μ-calculus formulae $\Theta \subseteq CL(\phi)$, the pair $\langle \phi, \Theta\rangle$ is called a *second-order assertion*. Define $\sigma \models \langle \phi, \Theta\rangle$ if $s \models \phi$ for $s \in \sigma$ provided $end(s) \models \Theta$. Intuitively, a location satisfies a second-order assertion $\langle \phi, \Theta\rangle$ if its instances under the given contextual assumptions Θ satisfy the μ-calculus formula ϕ.

We now describe the local model checking algorithm in [13]. Let $\mathbb{P} = \langle P_0, \ldots, P_n\rangle$ be a context-free process system with $P_i = \langle \Sigma_i, \underline{n}, Act, \rightarrow_i, \delta_i, \epsilon_i\rangle$ for $0 \leq i \leq n$, $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e\rangle$ its process graph, and ϕ a μ-calculus formula. A *sequent* is of the form $s \vdash \phi$ or $\sigma \vdash \langle \phi, \Theta\rangle$. We call the former *first-order* and the latter *second-order* sequents respectively. Let Ω be a set of sequents and ω a sequent. An *inference rule* is represented as $\frac{\Omega}{\omega}$. The sequents in Ω and ω are the *premises* and *conclusion* of the inference rule respectively. For clarity, we write $\frac{\omega}{\omega'}$ and $\frac{\omega_0 \qquad \omega_1}{\omega'}$ for $\frac{\{\omega\}}{\omega'}$ and $\frac{\{\omega_0, \omega_1\}}{\omega'}$ respectively. A *proof* is a tree rooted at a given sequent and constructed according to the inference rules in Figure 1. The start rule first guesses initial contextual assumptions Θ for the given property ϕ in the second-order assertion $\langle \phi, \Theta\rangle$. The assumptions Θ must be satisfied after the invocation of the main process. Similarly, contextual assumptions are chosen in modality rules. There are only finitely many possible contextual assumptions for any μ-calculus formula ϕ because $CL(\phi)$ is finite.

To show a location satisfies a conjunction in a second-order assertion, one proves that both conjuncts are satisfied under the same contextual assumptions. Symmetrically, a disjunct under the same assumptions in a second-order assertion must be satisfied in a disjunction. For fixed points, the inference rules simply unroll the formula. The unrolling need be justified on the leaves of the proof (Definition 6 (vi)). A (d, r)-*proof* is a proof which applies the fixed point and modality rules at most d and r times along any path from the root to a leaf respectively.

Definition 6. *A leaf of a proof is* successful *if it has one of the following forms.*

(i) $s_e \vdash \mathtt{tt}$*;*
(ii) $s_e \vdash [A]\phi$*;*
(iii) $\sigma \vdash \langle \mathtt{tt}, \Theta \rangle$*;*
(iv) $\sigma \vdash \langle [A]\phi, \Theta \rangle$ *where* $\sigma \in \Sigma_i$ *is not an end location and there is no* σ' *with* $\sigma \xrightarrow{a}_i \sigma'$ *for any* $a \in A$*, or* $\sigma \xrightarrow{j}_i \sigma'$*;*
(v) $\epsilon \vdash \langle \phi, \Theta \rangle$ *and* $\phi \in \Theta$*; or*
(vi) $\sigma \vdash \langle \phi(\nu X.\psi), \Theta \rangle$ *where* $\phi(\nu X.\psi) \in CL(\nu X.\psi)$ *and the same sequent re-occurs on the path from the root to itself.*

A finite proof is *successful* if all its leaves are successful. A sequent is *derivable* if there is a successful proof rooted in the sequent. A formula ϕ is *derivable* if the sequent $s_0 \vdash \phi$ is derivable. The following theorem shows the inference rules in Figure 1 are sound and complete.

Theorem 1. *([13]) An alternation-free* μ*-calculus formula* ϕ *is derivable for a context-free process system* $\mathbb{P}$ *iff it is satisfied in* $\mathbb{P}$*.*

An exemplary run of the local model checking algorithm is shown in Figure 2. In the figure, a simple context-free process P with an infinite number of states is considered. After performing the action a, the process P can either call itself recursively or terminates by executing b. We verify that the start state s_0 satisfies $\nu X.[a, b]X$ by the successful proof in the figure. Observe there are two nondeterministic choices of contextual assumptions in the applications of start and modality rules. They happen to be the same in the proof.

3 Proof search by SAT

Suppose $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ is a context-free process system with $P_i = \langle \Sigma_i, \underline{n}, Act, \rightarrow_i, \delta_i, \epsilon_i \rangle$ for $0 \leq i \leq n$, and $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$ its process graph. Since the number of locations in $\Sigma_0 \cup \cdots \cup \Sigma_n$ is finite, we can use a Boolean vector of size $\lg(\sum_{i=0}^{n} |\Sigma_i|)$ to encode locations. The Boolean vector representing the location σ is denoted by $\overline{\sigma}$. Moreover, we assume a fixed linear order on $CL(\phi)$ for the μ-calculus formula ϕ and denote the i-th formula in $CL(\phi)$ by ϕ_i. Any subset Θ of $CL(\phi)$ can hence be encoded by a Boolean vector $\bar{z}$ of size $|CL(\phi)|$ such that $\bar{z}[i] = \mathtt{tt}$ if and only if $\phi_i \in \Theta$. The Boolean vector representing the subset $\Theta \subseteq CL(\phi)$ is denoted by $\overline{\Theta}$.

Let ϕ be an alternation-free μ-calculus formula without universal modal operators. Figure 3 gives our encoding of proof search in the local model checking algorithm. In the figure, the Boolean variable vectors $\bar{u}, \bar{v}, \bar{w}$ encode locations and are of size $\lg(\Sigma_{i=0}^{n}|\Sigma_i|)$. The Boolean variable vectors $\bar{z}$ and $\bar{z}'$ encode a subset of $CL(\phi)$ and hence of size $|CL(\phi)|$. The list Γ consists of triples of the form $(\bar{u}, \phi, \bar{z})$. It records all second-order sequents $\sigma \vdash \langle \phi, \Theta \rangle$ from the root to the current sequent. The notation $(\bar{u}, \phi, \bar{z}) :: \Gamma$ represents the list whose elements are $(\bar{u}, \phi, \bar{z})$ followed by those in Γ. $|\Gamma|$ denotes the size of the list Γ. Intuitively,

Start rule

$$\frac{\{\delta_0 \vdash \langle \phi, \Theta \rangle\} \cup \{s_e \vdash \theta \mid \theta \in \Theta\}}{s_0 \vdash \phi}$$

End rules

$$\frac{s_e \vdash \phi_0 \qquad s_e \vdash \phi_1}{s_e \vdash \phi_0 \wedge \phi_1}$$

$$\frac{s_e \vdash \phi_0}{s_e \vdash \phi_0 \vee \phi_1} \qquad \frac{s_e \vdash \phi_1}{s_e \vdash \phi_0 \vee \phi_1}$$

$$\frac{s_e \vdash \phi[\mathsf{tt}/X]}{s_e \vdash \nu X.\phi} \qquad \frac{s_e \vdash \phi[\mathsf{ff}/X]}{s_e \vdash \mu X.\phi}$$

Conjunction and disjunction rules

$$\frac{\sigma \vdash \langle \phi_0, \Theta \rangle \qquad \sigma \vdash \langle \phi_1, \Theta \rangle}{\sigma \vdash \langle \phi_0 \wedge \phi_1, \Theta \rangle}$$

$$\frac{\sigma \vdash \langle \phi_0, \Theta \rangle}{\sigma \vdash \langle \phi_0 \vee \phi_1, \Theta \rangle} \qquad \frac{\sigma \vdash \langle \phi_1, \Theta \rangle}{\sigma \vdash \langle \phi_0 \vee \phi_1, \Theta \rangle}$$

Fixed point rule

$$\frac{\sigma \vdash \langle \phi[\nu X.\phi/X], \Theta \rangle}{\sigma \vdash \langle \nu X.\phi, \Theta \rangle} \qquad \frac{\sigma \vdash \langle \phi[\mu X.\phi/X], \Theta \rangle}{\sigma \vdash \langle \mu X.\phi, \Theta \rangle}$$

Modality rules

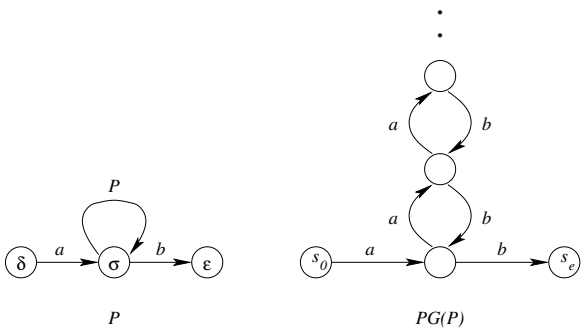
$$\frac{\{\sigma' \vdash \langle \phi, \Theta \rangle \mid a \in A, \sigma \xrightarrow{a}_i \sigma'\} \cup \bigcup_{j:\sigma \xrightarrow{j}_i \sigma'} (\{\delta_j \vdash \langle [A]\phi, \Psi_j \rangle\} \cup \{\sigma' \vdash \langle \psi, \Theta \rangle \mid \psi \in \Psi_j\})}{\sigma \vdash \langle [A]\phi, \Theta \rangle}$$

$$\frac{\sigma' \vdash \langle \phi, \Theta \rangle}{\sigma \vdash \langle \langle A \rangle \phi, \Theta \rangle}\ a \in A, \sigma \xrightarrow{a}_i \sigma' \qquad \frac{\{\delta_j \vdash \langle \langle A \rangle \phi, \Psi \rangle\} \cup \{\sigma' \vdash \langle \psi, \Theta \rangle \mid \psi \in \Psi\}}{\sigma \vdash \langle \langle A \rangle \phi, \Theta \rangle}\ \sigma \xrightarrow{j}_i \sigma'$$

Weakening rule

$$\frac{\sigma \vdash \langle \phi, \Theta' \rangle}{\sigma \vdash \langle \phi, \Theta \rangle}\ (\Theta' \subseteq \Theta)$$

Fig. 1. Local Model Checking Algorithm



$$\text{(Modality)}\ \frac{\text{(Modality)}\ \dfrac{\epsilon \vdash \langle M, \{M\}\rangle \qquad \text{(Modality)}\ \dfrac{\sigma \vdash \langle M, \{M\}\rangle}{\delta \vdash \langle N, \{M\}\rangle}}{\text{(Fixed point)}\ \dfrac{\sigma \vdash \langle N, \{M\}\rangle}{\text{(Modality)}\ \dfrac{\sigma \vdash \langle M, \{M\}\rangle}{\text{(Fixed point)}\ \dfrac{\delta \vdash \langle N, \{M\}\rangle}{\text{(Start)}\ \dfrac{\delta \vdash \langle M, \{M\}\rangle}{s_0 \vdash M}}}}} \qquad \text{(End)}\ \frac{s_e \vdash [a,b]\mathtt{tt}}{s_e \vdash M}$$

where $M = \nu X.[a,b]X$ and $N = [a,b]\nu X.[a,b]X$

Fig. 2. An Example of Local Model Checking

the idea is to construct a Boolean formula whose satisfiability is equivalent to a bounded proof. Another Boolean formula whose unsatisfiability is equivalent to completion will be built later. The model checking problem for context-free processes is therefore reduced to Boolean (un)satisfiability.

The following lemma states that our encoding of the end rules is correct.

Lemma 1. *$s_e \vdash \phi'$ is derivable iff $\lambda(s_e, \phi') = \mathtt{tt}$.*

To encode the derivation of the second-order sequent $\sigma \vdash \langle\langle A\rangle\phi', \Theta\rangle$, we let SAT solvers choose the contextual assumption Ψ in modality rules. The new assumption Ψ is represented by $\bar{z}'$ in the Boolean formula Π ($\bar{u}$, A, ϕ', $\bar{z}$, Γ, d, r). Given an assignment ρ, the valuation $[\![\bar{u}]\!]_\rho$ maps a Boolean variable vector $\bar{u}$ to a Boolean vector. The function $\chi(\bar{u}, A, \bar{v})$ in $\pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r)$ is $\mathtt{tt}$ with respect to an assignment ρ if and only if $[\![\bar{u}]\!]_\rho = \overline{\sigma}$, $[\![\bar{v}]\!]_\rho = \overline{\sigma}'$, and $\sigma \xrightarrow{a}_i \sigma'$ for some i and $a \in A$. Similarly, $\zeta(\bar{u}, \bar{v}, \bar{w}) = \mathtt{tt}$ with respect to an assignment ρ if and only if $[\![\bar{u}]\!]_\rho = \overline{\sigma}$, $[\![\bar{v}]\!]_\rho = \delta_j$, $[\![\bar{w}]\!]_\rho = \overline{\sigma}'$, and $\sigma \xrightarrow{j}_i \sigma'$ for some i. For the derivation of the sequent $\sigma \vdash \langle \eta X.\phi', \Theta\rangle$ where η is either the least or greatest fixed point operator, we simply unroll the formula and let $Successful\_leaf(\bar{u}, \phi', \bar{z}, \Gamma)$ check whether the sequent is successful or not. The variable c_i's in fixed point and modality rules are called *expansion variables*. Intuitively, c_i's indicate a proof

Auxiliaries

$$\Upsilon_0(\bar{u}) \triangleq \bigvee_{i=0}^{n}(\overline{\epsilon}_i = \bar{u}) \qquad \Upsilon_1(\phi_k, \bar{z}) \triangleq \bar{z}[k]$$

$$\Omega_0(\bar{u}, \phi', \bar{z}, \Gamma) \triangleq \bigvee_{k=0}^{|\Gamma|-1}(\bar{u}, \phi', \bar{z}) = \Gamma[k]$$

$$Not\_leaf(\bar{u}, \phi', \bar{z}, \Gamma) \triangleq \neg\Upsilon_0(\bar{u}) \wedge \neg\Omega_0(\bar{u}, \phi', \bar{z}, \Gamma)$$

$$Successful\_leaf(\bar{u}, \phi', \bar{z}, \Gamma) \triangleq \begin{cases} (\Upsilon_0(\bar{u}) \wedge \Upsilon_1(\phi', \bar{z})) \vee \Omega_0(\bar{u}, \phi', \bar{z}, \Gamma) & \text{if } \phi' \in CL(\nu X.\psi) \\ \Upsilon_0(\bar{u}) \wedge \Upsilon_1(\phi', \bar{z}) & \text{if } \phi' \notin CL(\nu X.\psi) \end{cases}$$

Start rule

$$\alpha(\phi, d, r) \triangleq \bar{u} = \bar{\delta}_0 \wedge \Lambda(\bar{u}, \phi, \bar{z}, [], d, r) \wedge \bigwedge_{k=0}^{|\bar{z}|-1}(\bar{z}[k] \Rightarrow \lambda(s_e, \phi_k))$$

where $\bar{z}$: a vector of fresh Boolean variables of size $|CL(\phi)|$

End rules

$$\lambda(s_e, \phi'_0 \wedge \phi'_1) \triangleq \lambda(s_e, \phi'_0) \wedge \lambda(s_e, \phi'_1) \qquad \lambda(s_e, \phi'_0 \vee \phi'_1) \triangleq \lambda(s_e, \phi'_0) \vee \lambda(s_e, \phi'_1)$$

$$\lambda(s_e, \nu X.\phi') \triangleq \lambda(s_e, \phi'[\mathsf{tt}/X]) \qquad \lambda(s_e, \mu X.\phi') \triangleq \lambda(s_e, \phi'[\mathsf{ff}/X])$$

$$\lambda(s_e, \langle a\rangle\phi') \triangleq \mathsf{ff} \qquad \lambda(s_e, \mathsf{tt}) \triangleq \mathsf{tt} \qquad \lambda(s_e, \mathsf{ff}) \triangleq \mathsf{ff}$$

Conjunction and disjunction rules

$$\Lambda(\bar{u}, \phi'_0 \wedge \phi'_1, \bar{z}, \Gamma, d, r) \triangleq$$
$$Successful\_leaf(\bar{u}, \phi'_0 \wedge \phi'_1, \bar{z}, \Gamma) \vee (\Lambda(\bar{u}, \phi'_0, \bar{z}, \Gamma', d, r) \wedge \Lambda(\bar{u}, \phi'_1, \bar{z}, \Gamma', d, r))$$
where $\Gamma' = (\bar{u}, \phi'_0 \wedge \phi'_1, \bar{z}) :: \Gamma$

$$\Lambda(\bar{u}, \phi'_0 \vee \phi'_1, \bar{z}, \Gamma, d, r) \triangleq$$
$$Successful\_leaf(\bar{u}, \phi'_0 \vee \phi'_1, \bar{z}, \Gamma) \vee (\Lambda(\bar{u}, \phi'_0, \bar{z}, \Gamma', d, r) \vee \Lambda(\bar{u}, \phi'_1, \bar{z}, \Gamma', d, r))$$
where $\Gamma' = (\bar{u}, \phi'_0 \vee \phi'_1, \bar{z}) :: \Gamma$

Fixed point rule

$$\Lambda(\bar{u}, \eta X.\phi', \bar{z}, \Gamma, d, r) \triangleq$$
$$\begin{cases} Successful\_leaf(\bar{u}, \eta X.\phi', \bar{z}, \Gamma) \vee (Not\_leaf(\bar{u}, \eta X.\phi', \bar{z}, \Gamma) \wedge c_i) & \text{if } d = 0 \\ Successful\_leaf(\bar{u}, \eta X.\phi', \bar{z}, \Gamma) \vee \Lambda(\bar{u}, \phi'[\eta X.\phi'/X], \bar{z}, \Gamma', d-1, r)) & \text{if } d > 0 \end{cases}$$
where c_i : a fresh Boolean variable and $\Gamma' = (\bar{u}, \eta X.\phi', \bar{z}) :: \Gamma$

Modality rules

$$\Lambda(\bar{u}, \langle A\rangle\phi', \bar{z}, \Gamma, d, r) \triangleq$$
$$\begin{cases} Successful\_leaf(\bar{u}, \langle A\rangle\phi', \bar{z}, \Gamma) \vee (Not\_leaf(\bar{u}, \langle A\rangle\phi', \bar{z}, \Gamma) \wedge c_i) & \text{if } r = 0 \\ Successful\_leaf(\bar{u}, \langle A\rangle\phi', \bar{z}, \Gamma) \vee \pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r) \vee \Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r) & \text{if } r > 0 \end{cases}$$
where c_i : a fresh Boolean variable

$$\pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r) \triangleq \chi(\bar{u}, A, \bar{v}) \wedge \Lambda(\bar{v}, \phi', \bar{z}, \Gamma', d, r)$$

$$\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r) \triangleq$$
$$\zeta(\bar{u}, \bar{v}, \bar{w}) \wedge \Lambda(\bar{v}, \langle A\rangle\phi', \bar{z}', \Gamma', d, r-1) \wedge \bigwedge_{k=0}^{|\bar{z}'|-1}(\bar{z}'[k] \Rightarrow \Lambda(\bar{w}, \phi_k, \bar{z}, \Gamma', d, r-1))$$
$\bar{v}, \bar{w}$: vectors of fresh Boolean variables of size $\lg(\sum_{i=0}^{n}|\Sigma_i|)$
where $\bar{z}'$: a vector of fresh Boolean variables of size $|CL(\phi)|$
$\Gamma' = (\bar{u}, \langle A\rangle\phi', \bar{z}) :: \Gamma$

Atomic rules

$$\Lambda(\bar{u}, \mathsf{tt}, \bar{z}, \Gamma, d, r) \triangleq \mathsf{tt} \qquad \Lambda(\bar{u}, \mathsf{ff}, \bar{z}, \Gamma, d, r) \triangleq \mathsf{ff}$$

Fig. 3. Proof Search in Boolean Satisfiability

needs to apply more fixed point and modality rules. The following lemma shows that a satisfying assignment ρ for $\Lambda(\bar{u}, \phi', \bar{z}, [], d, r) \wedge \bigwedge_{i=0}^{l} \neg c_i$ corresponds to a successful proof for the sequent $\sigma \vdash \langle \phi', \Theta \rangle$ with $\overline{\sigma} = [\![\bar{u}]\!]_\rho$ and $\overline{\Theta} = [\![\bar{z}]\!]_\rho$.

Lemma 2. *Let $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ be a context-free process system, $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$ its process graph, and $c_0, \ldots, c_l$ the expansion variables in $\Lambda(\bar{\delta}_0, \phi', \bar{z}, [], d, r)$ with $d, r \in \mathbb{N}$. If $\bar{u} = \bar{\delta}_0 \wedge \Lambda(\bar{u}, \phi', \bar{z}, [], d, r) \wedge \bigwedge_{i=0}^{l} \neg c_i$ is satisfied by the assignment ρ, there is a successful proof for $\delta_0 \vdash \langle \phi', \Theta \rangle$ with $[\![\bar{z}]\!]_\rho = \overline{\Theta}$.*

On the other hand, the formula $\Lambda(\bar{u}, \phi', \bar{z}, [], d, r) \wedge \bigwedge_{i=0}^{l} \neg c_i$ can be satisfied by the assignment ρ if a successful (d, r)-proof for the second-order sequent $\sigma \vdash \langle \phi', \Theta \rangle$ with $\overline{\sigma} = [\![\bar{u}]\!]_\rho$ and $\overline{\Theta} = [\![\bar{z}]\!]_\rho$ exists.

Lemma 3. *Let $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ be a context-free process system, $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$ its process graph, and $c_0, \ldots, c_l$ the expansion variables in $\bar{u} = \overline{\sigma} \wedge \Lambda(\bar{u}, \phi', \bar{z}, [], d, r)$ where $d, r \in \mathbb{N}$. If there is a successful (d, r)-proof for $\sigma \vdash \langle \phi', \Theta \rangle$, then there is a satisfying assignment ρ for $\bar{u} = \overline{\sigma} \wedge \Lambda(\bar{u}, \phi', \bar{z}, [], d, r) \wedge \bigwedge_{i=0}^{l} \neg c_i$ such that $[\![\bar{u}]\!]_\rho = \overline{\sigma}$ and $[\![\bar{z}]\!]_\rho = \overline{\Theta}$.*

By Lemma 2 and 3, a satisfying assignment ρ for $\Lambda(\bar{u}, \phi', \bar{z}, [], d, r)$ and a successful (d, r)-proof for the second-order sequent $\sigma \vdash \langle \phi', \Theta \rangle$ are related by $[\![\bar{u}]\!]_\rho = \overline{\sigma}$ and $[\![\bar{z}]\!]_\rho = \overline{\Theta}$. If the start rule is furthermore taken into consideration, we have the following theorem.

Theorem 2. *Let $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ be a context-free process system, $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$ its process graph, and $c_0, \ldots, c_l$ the expansion variables in $\bar{u} = \alpha(\phi, d, r)$ with $d, r \in \mathbb{N}$. Define $\Xi_-(\phi, d, r) \stackrel{\triangle}{=} \alpha(\phi, d, r) \wedge \bigwedge_{i=0}^{l} \neg c_i$.*

(i) If $\Xi_-(\phi, d, r)$ is satisfiable, then there is a successful proof for $s_0 \vdash \phi$.
(ii) If there is a successful (d, r)-proof for $s_0 \vdash \phi$, then $\Xi_-(\phi, d, r)$ is satisfiable.

Given two integers d and r, Theorem 2 shows that a successful (d, r)-proof for a second-order sequent exists exactly when the Boolean formula $\Xi_-(\phi, d, r)$ is satisfiable. But we have no information when the Boolean formula is unsatisfiable. Particularly, we do not know if any (d, r)-proof exists for larger d or r. The following lemma states that we need not continue the proof search when a similar formula is unsatisfiable.

Lemma 4. *If there is a successful (d', r')-proof for $\sigma \vdash \langle \phi', \Theta \rangle$ with $d' > d$ or $r' > r$, and $c_0, \ldots, c_l$ are the expansion variables in $\Lambda(\bar{u}, \phi', \bar{z}, [], d, r)$, then there is a satisfying assignment ρ for $\bar{u} = \overline{\sigma} \wedge \Lambda(\bar{u}, \phi', \bar{z}, [], d, r) \wedge \bigwedge_{i=0}^{l} c_i$ with $[\![\bar{z}]\!]_\rho = \overline{\Theta}$.*

By considering the start location and adding the start rule, we have the following criteria for the completion of proof search.

Theorem 3. *Let $\mathbb{P} = \langle P_0, \ldots, P_n \rangle$ be a context-free process system, $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$ its process graph, and $c_0, \ldots, c_l$ the expansion variables in $\alpha(\phi, d, r)$ with $d, r \in \mathbb{N}$. Define $\Xi_+(\phi, d, r) \stackrel{\triangle}{=} \alpha(\phi, d, r)$. If there is a successful (d', r')-proof for $s_0 \vdash \phi$ with $d' > d$ or $r' > r$, then there is an assignment ρ satisfying $\Xi_+(\phi, d, r)$.*

The unsatisfiability of the formula $\Xi_+(\phi, d, r)$ corresponds to the absence of proof in local model checking. Again, one must show that $\Xi_+(\phi, d, r)$ is eventually unsatisfiable if the property fails at the initial state. Our technical results are now summarized in the following two theorems.

Theorem 4. *(Soundness and completeness)*

1. *If $\Xi_-(\phi, d, r)$ is satisfiable, then there is a successful (d, r)-proof for $s_0 \vdash \phi$.*
2. *If there is a successful (d, r)-proof for $s_0 \vdash \phi$, then $\Xi_-(\phi, d, r)$ is satisfiable.*

Theorem 5. *(Completion and termination)*

1. *If $\Xi_+(\phi, d, r)$ is unsatisfiable, then there is no successful (d', r')-proof for $s_0 \vdash \phi$ with $d' > d$ or $r' > r$.*
2. *If there is no successful proof for $s_0 \vdash \phi$, then there are some d and r such that $\Xi_+(\phi, d, r)$ is unsatisfiable.*[4]

4 Algorithm

Our SAT-based model checking algorithm for context-free processes is shown in Figure 4. The algorithm first computes the negative normal form of the negation of the given property. Proofs of refutation and completion are checked incrementally. If a proof of the negated property is found, the algorithm reports an error. If, on the other hand, the completion criteria is satisfied, it reports success.

Given a context-free process system $\mathbb{P}$ and $PG(\mathbb{P}) = \langle S, Act, \rightarrow, s_0, s_e \rangle$
Let ψ be a universal alternation-free μ-calculus formula
Let ϕ be the negative normal form of $\neg\psi$
$d \leftarrow 0$
loop
  if $\Xi_-(\phi, d, d)$ is satisfiable **then**
    return "$s_0 \nvdash \psi$"
  if $\Xi_+(\phi, d, d)$ is unsatisfiable **then**
    return "$s_0 \vdash \psi$"
  $d \leftarrow d + 1$
end

Fig. 4. Model Checking Algorithm

By Theorem 4, a proof of the negated property can always be found should it exist. Otherwise, the algorithm checks whether the search should continue by Theorem 5 (1). If a proof is possible, the algorithm increments the bounds and repeats. It will terminate if there is no proof (Theorem 5 (2)). Applying Theorem 1, we have the following theorem.

[4] Intuitively, $Successful\_leaf(\bar{u}, \phi', \bar{z}, \Gamma)$ is unsatisfiable when there is no successful proof. But $Not\_leaf(\bar{u}, \phi', \bar{z}, \Gamma)$ will become unsatisfiable eventually.

Theorem 6. *The algorithm in Figure 4 is correct. Namely, it has the following properties.*

- *It always terminates.*
- *It reports "$s_0 \vdash \phi$" if and only if $s_0 \models \phi$.*

5 Experiments

5.1 Implementation

We have implemented our SAT-based model checking algorithm for context-free processes in Objective Caml [14]. Given a context-free process and a formula in the universal fragment of alternation-free μ-calculus, our implementation creates instances of the Boolean satisfiability and solves them using MiniSat [10].

Most of our implementation is straightforward, but care must be taken for modality rules in Figure 3. The formula $\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r)$ presumes process invocation for any location represented by $\bar{u}$. Hence it may construct $\Lambda(\bar{w},\ \phi_k,\ \bar{z},\ \Gamma',\ d,\ r-1)$ for $1 \leq k \leq |CL(\phi)|$ unnecessarily. Precisely, the formulae $\Lambda(\bar{w}, \phi_k, \bar{z}, \Gamma', d, r-1)$ are not needed when $\zeta(\bar{u}, \bar{v}, \bar{w})$ is unsatisfiable.

To avoid creating unneeded formulae in $\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r)$, we compute the set of locations reachable by r transitions from the start location of the main process P_0. If there is no location in the set that may invoke other processes, $\zeta(\bar{u}, \bar{v}, \bar{w})$ is unsatisfiable and $\Lambda(\bar{w}, \phi_k, \bar{z}, \Gamma', d, r-1)$ can be omitted. More formally, let δ be the start state of P_0. Define $\Delta_0 \stackrel{\triangle}{=} \{\delta\}$ and $\Delta_{i+1} \stackrel{\triangle}{=} \{\sigma' : \sigma \xrightarrow{\alpha} \sigma' \text{ for some } \sigma \in \Delta_i\}$. Modify $\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r)$ as follows.

$$\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r) \stackrel{\triangle}{=}$$
$$\begin{cases} \mathsf{ff} & \text{if } \forall \sigma \in \Delta_r, i \in \underline{n}.\sigma \stackrel{i}{\not\rightarrow} \\ \zeta(\bar{u}, \bar{v}, \bar{w}) \wedge \Lambda(\bar{v}, \langle A \rangle \phi', \bar{z}', \Gamma', d, r-1) \wedge & \\ \quad \bigwedge_{k=0}^{|\bar{z}'|-1} (\bar{z}'[k] \Rightarrow \Lambda(\bar{w}, \phi_k, \bar{z}, \Gamma', d, r-1)) & \text{otherwise} \end{cases}$$

Intuitively, Δ_i contains locations reachable from the start location δ_0 with i transitions. If none of the locations in Δ_i performs process invocation, it is unnecessary to expand $\Pi(\bar{u}, A, \phi', \bar{z}, \Gamma, d, r)$ further. The simple modification avoids creating unneeded formulae and can improve the performance significantly.

5.2 Experimental results

We compare our SAT-based algorithm with a BDD-based algorithm on randomly generated context-free process systems in [12]. A location may be sequential, branching, and looping with probabilities 0.6, 0.2, and 0.2 respectively. Moreover, context-free processes may be called with probability 0.2 on the sequential location.

Two properties are checked against the randomly generated system. The liveness property checks if a random location σ of the main process is reachable on all paths:

$\mu X.([A]\mathtt{tt} \vee [Act]X)$, where A is the specific set of actions for σ.

The safety property checks if a random process P_i is never called on all paths:

$\nu X.([A]\mathtt{ff} \wedge [Act]X)$, where A is the specific set of actions for δ_i.

#process/ avg. #location	*liveness*			*safety*		
	ans.	BDD (sec.)	SAT (sec.)	ans.	BDD (sec.)	SAT (sec.)
3/1k	*No*	0.02	0.03	*Yes*	0.02	0.12
4/2k	*No*	0.10	0.14	*No*	0.12	0.24
5/4k	*No*	0.09	0.16	*Yes*	0.18	1.70
6/8k	*No*	0.38	0.54	*Yes*	0.54	7.78
7/16k	*No*	1.51	1.44	*No*	2.57	3.54
8/32k	*No*	19.07	7.07	*No*	35.22	7.61
9/64k	*No*	23.44	9.21	*No*	46.89	8.31
10/128k	*No*	O/M	49.17	*No*	O/M	55.11

(measured in a 1.6 GHz Intel machine with 512Mb memory)

Table 1. Performance Comparison

In table 1, the performance data of our SAT-based algorithm and the BDD-based algorithm are shown. Our SAT-based algorithm outperforms the BDD-based algorithm for larger context-free process systems (8/32k, 9/64k, 10/128k) on both properties. For the largest case (10/128k), our SAT-based algorithm is capable of finding errors while the BDD-based algorithm running out of memory.

Our experiments show that the execution time of the BDD-based algorithm increases consistently with the sizes of the context-free process systems. On the other hand, SAT-based algorithms are known to be very efficient in bug detection [4, 3]. Indeed, our SAT-based algorithm takes more time in proving the safety property for the 6/8k system than falsifying it for the 7/16k system.

Table 2 gives time distribution of our implementation. We measure the time

- for reading the input file and model building;
- for creating instances of Boolean satisfiability; and
- for solving the instances.

When the property does not hold, our implementation spends most of the time reading the input and building models. For the verification of the safety property on 5/4k, 6/8k, and 7/16k, the majority of time is used for creating instances. This suggests further improvement could still be possible for our implementation.

6 Conclusion

Because of its capacity and scalability, SAT solvers have found many applications in hardware verification. On the other hand, alternative computational models

#process/ avg. #location	*liveness*			*safety*		
	read	create	solve	read	create	solve
3/1k	0.02	0.01	<0.01	0.05	0.07	<0.01
4/2k	0.14	0.00	<0.01	0.16	0.08	0.01
5/4k	0.15	0.01	<0.01	0.18	1.27	0.24
6/8k	0.52	0.01	<0.01	0.58	4.56	2.63
7/16k	1.42	0.01	<0.01	1.53	1.68	0.32
8/32k	7.03	0.03	<0.01	6.94	0.59	0.06
9/64k	9.17	0.03	<0.01	8.21	0.08	0.01
10/128k	48.69	0.39	0.08	50.78	4.04	0.35

(execution time in seconds)

Table 2. Profiling data

have shown their promises in software verification. In this paper, a complete SAT-based model checking algorithm is developed to take advantages of SAT solvers and context-free processes. By combining the scalability of SAT solvers and the succinctness of context-free processes, the proposed algorithm could potentially analyze more programs. To the best of our knowledge, this is the first SAT-based model checking algorithm for context-free processes.

In comparison with other SAT-based techniques, our proof-theoretic approach is very general. Instead of exploiting characteristics in specification logics or computational models, we reduce the proof search in local model checking algorithms to the satisfiability problem. The same idea is used to develop an SAT-based model checking algorithm for finite-state models in [18]. The present work demonstrates the applicability of our approach even for context-free processes.

In the experiments, we show that our SAT-based algorithm outperforms in some cases. We would like to conduct more experiments to support our preliminary findings in the future.

Although we believe context-free processes should be sufficient for program analysis, a scientific study will be very welcome. Finally, it would be interesting to see if our technique can be applied to other infinite-state models.

Acknowledgment. The authors would like to thank anonymous reviewers for their comments and suggestions in revising the paper.

References

1. Awedh, M., Somenzi, F.: Proving more properties with bounded model checking. In Alur, R., Peled, D.A., eds.: CAV. Volume 3114 of LNCS., Springer Verlag (2004) 96–108
2. Ball, T., Rajamani, S.: Bebop: A symbolic model checker for boolean programs. In Havelund, K., Penix, J., Visser, W., eds.: SPIN 2000 Workshop on Model Checking of Software. Volume 1885 of LNCS., Springer-Verlag (2000)
3. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: DAC, ACM Press (1999) 317–320

4. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In Cleaveland, W.R., ed.: TACAS. Volume 1579 of LNCS., Springer-Verlag (1999) 193–207
5. Boujjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Applications to model checking. In Mazurkiewicz, A.W., Winkowski, J., eds.: CONCUR. Volume 1243 of LNCS., Springer-Verlag (1997) 135–150
6. Burkart, O., Steffen, B.: Model checking for context-free processes. In Cleaveland, R., ed.: CONCUR. Volume 630 of LNCS., Springer-Verlag (1992) 123–137
7. Burkart, O., Esparza, J.: More infinite results. In Paun, G., Rozenberg, G., Salomaa, A., eds.: Current Trends in Theoretical Computer Science, Entering the 21th Century. World Scientific (2001) 480–503
8. Caucal, D., Monfort, R.: On the transition graphs of automata and grammars. In Möhring, R.H., ed.: Graph-Theoretic Concepts in Computer Science. Volume 484 of LNCS., Springer-Verlag (1990) 311–337
9. Christensen, S., Hüttel, H.: Decidability issues for infinite-state processes - a survey. Bulletin of the European Association for Theoretical Computer Science **51** (1993) 156–166
10. Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of LNCS., Springer (2003) 502–518
11. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In Emerson, E.A., Sistla, A.P., eds.: CAV. Volume 1855 of LNCS., Springer-Verlag (2000) 232–247
12. Esparza, J., Schwoon, S.: A BDD-based model checker for recursive programs. In Berry, G., Comon, H., Finkel, A., eds.: CAV. Volume 2102 of LNCS., Springer-Verlag (2001) 324–336
13. Hungar, H., Steffen, B.: Local model checking for context-free processes. Nordic Journal of Computing **1**(3) (1994) 364–385
14. Leroy, X.: The Objective Caml system: Documentation and user's manual (2000) With Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon.
15. McMillan, K.L.: Interpolation and sat-based model checking. In Jr., W.A.H., Somenzi, F., eds.: CAV. Volume 2725 of LNCS., Springer Verlag (2003) 1–13
16. Muller, D.E., Schupp, P.E.: The theory of ends, pushdown automata, and sceond-order logic. Theoretical Computer Science **37** (1985) 51–75
17. Walukiewicz, I.: Pushdown processes: Games and model-checking. Information and Computation **164**(2) (2001) 234–263
18. Wang, B.Y.: Proving $\forall\mu$-calculus properties with SAT-based model checking. In Wang, F., ed.: FORTE. Volume 3731 of LNCS., Springer-Verlag (2005) 113–127