Pushdown Processes: Parallel Composition and Model Checking

Olaf Burkart*
Lehrstuhl für Informatik II
RWTH Aachen
Ahornstraße 55
52056 Aachen
Germany

Bernhard Steffen†
Fakultät für Mathematik und Informatik
Universität Passau
Innstraße 33
Postfach 2540
94032 Passau
Germany

Keywords:

infinite-state systems, context-free processes, pushdown processes, parallel composition, model-checking

Abstract

In this paper we consider a strict generalization of context-free processes, the *pushdown processes*, and show that this class of processes is 1) closed under parallel composition with finite state systems, and can 2) be model checked by means of an elegant adaptation of the higher order model checker introduced in [BS92]. This shows the advantages of pushdown processes over context-free processes, which are not sufficiently general in order to support parallel composition.

*e-mail: burkart@informatik.rwth-aachen.de
†e-mail: steffen@fmi.uni-passau.de

1 Introduction

Over the past decade model-checking has emerged as a powerful tool for the automatic analysis of concurrent systems. Whereas model-checking for finite-state systems is well-established (cf. e.g. [EL86, CES86, Lar88, SW89, Cle90, CS92]), the theory for infinite systems is a current research topic. Bradfield and Stirling [Bra91, BS91] observed that tableaux-based model-checking covers general infinite-state systems. However, their method is not effective. Therefore much work has focused on *context-free processes*, a subclass of the infinite-state systems. In particular, in [BS92] an iterative model-checking algorithm is developed that decides the *alternation-free* part of the modal mu-calculus for *context-free* processes. Moreover, in [HS93] it is shown how this can be done using tableaux-based techniques.

Unfortunately, context-free processes are not closed under parallel composition. In fact, arbitrary Turing machines can be modelled by means of the parallel composition of two context-free processes. However, already the parallel composition of a context-free process with a finite state system is in general not context-free. Thus context-free processes do not support the construction of distributed systems.

In this paper we consider a strict generalization[1] of context-free processes, the *pushdown processes*, and show that this class of processes

- is closed under parallel composition with finite state systems. In fact, we are able to formulate a kind of expansion theorem in the sense of [Mil89].
- can be model checked by means of an elegant adaptation to pushdown automata of the higher order model checker introduced in [BS92].

Both results use a finite representation of pushdown processes in terms of pushdown automata. Intuitively, our Expansion Theorem 2.8 states that it is possible to express the parallel composition of a pushdown process and a finite process essentially as a product of the finite control of the underlying pushdown automaton and the finite state system, synchronized with a stack. It will be shown that the implied 'representation explosion' is no worse than the standard 'state explosion' for finite state systems.

Our iterative model-checking algorithm decides the *alternation-free* part of the modal mu-calculus for pushdown processes. As in the case of context-free processes, the point of our algorithm is to consider a *higher order* variant of the standard iterative model-checking techniques. It determines *property transformers* for the *fragments* of the pushdown automaton, which describe the set of formulas that are valid at the start state of a fragment relative to the set of formulas that are valid at the end states of the fragment. The number of end states of a fragment, which actually coincides with the number of states of the finite control of the underlying pushdown automaton, determines the arity of the corresponding property transformer. Thus it is only required to generalize the model checker for context-free processes, which only uses unary property transformers, in order to allow arbitrary arities. After the determination of these property transformers, the model-checking problem can be solved simply by checking whether the formula under consideration is a member of the set of formulas that results from applying

[1] There exist pushdown processes for which there is no equivalent context-free process up to bisimulation [CM90].

the property transformer associated with the initial fragment of the pushdown automaton to the set of formulas that are valid at the end states.

That model checking is decidable for pushdown processes is also a consequence of the fact that monadic second order logic, in which the modal mu-calculus can be embedded, is decidable for this class of processes (cf. [MS85]). New is the efficiency of our algorithm: only the size of the finite control and the size of the considered formula are critical (cf. Theorem 4.8).

The next section defines process graphs, our underlying process model, recalls pushdown automata, which characterize pushdown processes, and proves that pushdown processes are closed under parallel composition with finite-state systems. Subsequently, Section 3 presents our logic and Section 4 develops our model-checking algorithm. The final section contains conclusions and directions to future work, and the appendix provides details of the model checking algorithm.

2 Modelling Pushdown Processes

In this paper we model processes by means of process graphs:

Definition 2.1 (Process graph) *A* process graph *is a 5-tuple* $G = \langle \mathcal{S}, Act, \rightarrow, s_0, \langle e_1, \ldots, e_z \rangle \rangle$, *where* $\mathcal{S}$ *is a set of* states, Act *is a set of* actions *and* $\rightarrow \subseteq \mathcal{S} \times Act \times \mathcal{S}$ *is a* transition relation. *Furthermore,* $s_0 \in \mathcal{S}$ *is a distinguished element, the "start state", and* $\langle e_1, \ldots, e_z \rangle, e_i \in \mathcal{S}$ *is an ordered set of "end states", where* z *is called the* arity *of* G*, satisfying*

- s_0 *must be* originating*, i.e. there is no* $a \in Act, s' \in \mathcal{S}$ *with* $\langle s', a, s_0 \rangle \in \rightarrow$*; and*
- *each* e_i *must be* terminating*, i.e. there is no* $a \in Act, s' \in \mathcal{S}$ *with* $\langle e_i, a, s' \rangle \in \rightarrow$.

Intuitively, a process graph encodes the operational behaviour of a process. The set $\mathcal{S}$ represents the set of states the process may enter, Act the set of actions the process may perform and $\rightarrow$ the state transitions that may result upon execution of the actions. Note that "unrooted" process graphs, i.e. process graphs without specified start and end states, are just usual *labelled transition systems* (cf. [Mil89]).

A process that cannot perform any further actions is *terminated*, if the current state is an end state, and it is *deadlocked* otherwise. Moreover, we use $s \xrightarrow{a} s'$ in lieu of $\langle s, a, s' \rangle \in \rightarrow$ and we write $s \xrightarrow{a}$ when there is an s' such that $s \xrightarrow{a} s'$. If $s \xrightarrow{a} s'$ then we call s' an *a-derivative* of s. A process graph is *finitely branching* if the sets $\{ s' \mid s \xrightarrow{a} s', a \in Act \}$ are finite for all $s \in \mathcal{S}$ and it is said to be *finite-state* when $\mathcal{S}$ and Act are finite. Finally, the set of all process graphs is denoted by $\mathcal{G}$.

Process graphs can be combined to form more complex processes. Two operations are of special interest here: *sequential composition* and *nondeterministic choice*. Intuitively, the sequential composition $G \,;\, \langle G_1, \ldots, G_z \rangle$ of $z + 1$ process graphs behaves like G until an end state $e_i, i \in \{ 1, \ldots, z \}$ of G is reached. G has now terminated and the sequential composition continues to behave like G_i. The nondeterministic choice between two process graphs, $G_1 + G_2$, constitutes the process which behaves either like G_1 or like G_2.

Formally, denoting the process graph G, where the states s_1 and s_2 are identified, by $G[s_1 = s_2]$ and writing $\dot{\bigcup}$ for the disjoint union, sequential composition and nondeterministic choice are defined as follows:

Definition 2.2 (Sequential Composition)
Let $G_i = \langle \mathcal{S}^{(i)}, Act^{(i)}, \rightarrow^{(i)}, s_0^{(i)}, \langle e_1^{(i)}, \ldots, e_{z_i}^{(i)} \rangle \rangle, i \in I = \{0, \ldots, z_0\}$ *be process graphs. The* sequential composition $G_0 \; ; \; \langle G_1, \ldots, G_z \rangle$ *is the process graph*

$$\langle \dot{\bigcup_{i \in I}} \mathcal{S}^{(i)}, \bigcup_{i \in I} Act^{(i)}, \dot{\bigcup_{i \in I}} \rightarrow^{(i)}, s_0^{(0)}, \langle e_1^{(1)}, \ldots, e_{z_1}^{(1)}, \ldots, e_1^{(z_0)}, \ldots, e_{z_{z_0}}^{(z_0)} \rangle \rangle \, [e_i^{(0)} = s_0^{(i)}, i \in I]$$

Definition 2.3 (Nondeterministic Choice)
Let $G_i = \langle \mathcal{S}^{(i)}, Act^{(i)}, \rightarrow^{(i)}, s_0^{(i)}, \langle e_1^{(i)}, \ldots, e_{z_i}^{(i)} \rangle \rangle$, $i = 1, 2$ *be two process graphs. Then their* nondeterministic choice, $G_1 + G_2$, *is the process graph*

$$\langle \mathcal{S}^{(1)} \,\dot{\cup}\, \mathcal{S}^{(2)}, Act^{(1)} \cup Act^{(2)}, \rightarrow^{(1)} \,\dot{\cup}\, \rightarrow^{(2)}, s_0^{(1)}, \langle e_1^{(1)}, \ldots, e_{z_1}^{(1)}, e_1^{(2)}, \ldots, e_{z_2}^{(2)} \rangle \rangle \, [s_0^{(1)} = s_0^{(2)}]$$

As we will see in Section 4, our application allows us to use slightly specialized versions of these notions, which support a concise representation.

In order to capture when two processes are said to exhibit the same behaviour we use the well-known *bisimulation equivalence* [Mil89]:

Definition 2.4 *A binary relation R between processes is a* bisimulation *if whenever* $(P, Q) \in R$ *then for each* $a \in Act$

1. $P \xrightarrow{a} P'$ *implies* $\exists \, Q'. \; Q \xrightarrow{a} Q' \;\wedge\; (P', Q') \in R$
2. $Q \xrightarrow{a} Q'$ *implies* $\exists \, P'. \; P \xrightarrow{a} P' \;\wedge\; (P', Q') \in R$

Two processes P and Q are said to be *bisimulation equivalent* or *bisimilar*, written $P \sim Q$, if $(P, Q) \in R$ for some bisimulation R.

2.1 Pushdown Process Graphs

Pushdown processes are processes that can be (finitely) represented by means of classical pushdown automata (cf. [CM90]).

Definition 2.5 *A (nondeterministic)* pushdown automaton *(without final states[2]) is a 6-tuple* $\langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$, *where Q is a finite set of states, Act is a finite set of* actions, Γ *is finite set of* stack symbols, $q_1 \in Q$ *is the initial state,* $Z_1 \in \Gamma$ *is the initial stack symbol and δ is a mapping from* $Q \times Act \times \Gamma$ *to finite subsets of* $Q \times \Gamma^*$.

A pushdown automaton constitutes a process graph as follows:

Definition 2.6 *The* process graph $\mathtt{PG}(\mathcal{P})$ *of a pushdown automaton* $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ *with* $Q = \{q_1, \ldots, q_n\}$ *is the process graph*

$$\mathtt{PG}(\mathcal{P}) = \langle Q \times \Gamma^*, Act, \rightarrow_\delta, \langle q_1, Z_1 \rangle, \langle \langle q_1, \epsilon \rangle, \ldots, \langle q_n, \epsilon \rangle \rangle \rangle$$

where

$$\langle q, Z\gamma \rangle \xrightarrow{a}_\delta \langle q', \beta\gamma \rangle \quad \textit{iff} \quad \delta(q, a, Z) \ni \langle q', \beta \rangle \qquad \textit{for } q, q' \in Q, a \in Act, Z \in \Gamma \textit{ and } \beta, \gamma \in \Gamma^*$$

[2]Note the difference between end states and final states.

Note that $\mathtt{PG}(\mathcal{P})$ possesses an infinite number of states. However, the resulting process graph always is finitely branching[3] which is essential for our algorithm. – As usual we can restrict our attention to the set of states which are reachable from the start state.

As an example consider the pushdown automaton $\mathcal{A}$ of Figure 1. Figure 2 illustrates the corresponding infinite-state pushdown process $\mathtt{PG}(\mathcal{A})$. In [CM90] it is shown that there does not exist a context-free process which is equivalent to $\mathtt{PG}(\mathcal{A})$ up to bisimulation. This establishes that pushdown processes are strictly more expressive with respect to the bisimulation semantics than context-free processes.

$\mathcal{A} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ where

- $Q = \{ q_1, q_2, q_3, q_4 \}$
- $Act = \{ a, b, c, d \}$
- $\Gamma = \{ Z_1 \}$ and
- $\delta : Q \times Act \times \Gamma \to \mathfrak{P}_{fin}(Q \times \Gamma^*)$ with

$$\begin{array}{lcl}
\delta(q_1, a, Z_1) & = & \{ \langle q_1, Z_1 Z_1 \rangle \} \\
\delta(q_1, b, Z_1) & = & \{ \langle q_2, \epsilon \rangle \} \\
\delta(q_2, d, Z_1) & = & \{ \langle q_2, \epsilon \rangle \} \\
\delta(q_1, c, Z_1) & = & \{ \langle q_3, \epsilon \rangle \} \\
\delta(q_3, d, Z_1) & = & \{ \langle q_4, Z_1 \rangle \} \\
\delta(q_4, d, Z_1) & = & \{ \langle q_3, \epsilon \rangle \}
\end{array}$$

Figure 1: An Example Pushdown Automaton.

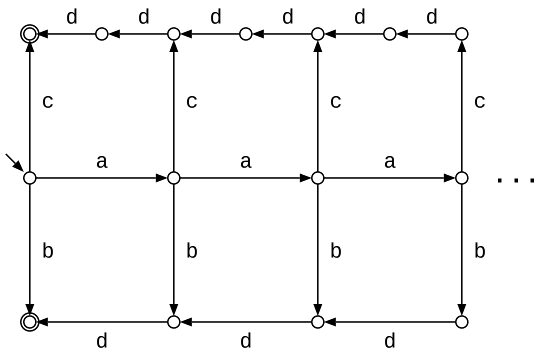

Figure 2: The Pushdown Process defined by the PDA $\mathcal{A}$.

2.2 Parallel Composition

We now introduce a binary CSP-like parallel operator $\parallel$ for process graphs (cf. [Hoa88]). Intuitively, the parallel composition $G_1 \parallel G_2$ of two process graphs requires the synchronization of the actions common to both component alphabets and allows the interleaving of the others.

Definition 2.7 *The* parallel composition $G = G_1 \parallel G_2$ *of two process graphs* $G_i = \langle \mathcal{S}_i, Act_i, \to_i, s_0^{(i)}, \{ e_1^{(i)}, \ldots, e_{z_i}^{(i)} \} \rangle, i = 1, 2$ *is the process graph*

$$G = \langle \mathcal{S}_1 \times \mathcal{S}_2, Act_1 \cup Act_2, \to_G, \langle s_0^{(1)}, s_0^{(2)} \rangle, \{ \langle e_i^{(1)}, e_j^{(2)} \rangle \mid 1 \le i \le z_1, 1 \le j \le z_j \} \rangle$$

where $\to_G$ *is defined as follows with* $s^{(1)}, s'^{(1)} \in \mathcal{S}_1, s^{(2)}, s'^{(2)} \in \mathcal{S}_2, a \in Act_1 \cup Act_2$*:*

- $\langle s^{(1)}, s^{(2)} \rangle \xrightarrow{a}_G \langle s'^{(1)}, s^{(2)} \rangle$ *iff* $s^{(1)} \xrightarrow{a}_1 s'^{(1)}$ *and* $a \notin Act_2$
- $\langle s^{(1)}, s^{(2)} \rangle \xrightarrow{a}_G \langle s^{(1)}, s'^{(2)} \rangle$ *iff* $s^{(2)} \xrightarrow{a}_1 s'^{(2)}$ *and* $a \notin Act_1$
- $\langle s^{(1)}, s^{(2)} \rangle \xrightarrow{a}_G \langle s'^{(1)}, s'^{(2)} \rangle$ *iff* $s^{(1)} \xrightarrow{a}_1 s'^{(1)}, s^{(2)} \xrightarrow{a}_1 s'^{(2)}$ *and* $a \in Act_1 \cap Act_2$

[3] $\mathtt{PG}(\mathcal{P})$ even is *boundedly branching*, i.e. there exists a $k \in \mathbb{N}$ with $|\{ s' \mid s \xrightarrow{a} s', a \in Act \}| \le k$ for all $s \in \mathcal{S}$.

Now we are able to formulate our main theorem which states that pushdown processes, in contrast to context-free processes, are closed under parallel composition with finite state processes. Thus pushdown processes seem to be the appropriate generalization of context-free processes for modelling distributed systems.

Theorem 2.8 (Expansion Theorem) *Given a pushdown automaton $\mathcal{P}$ and a finite process graph F, we can effectively construct a pushdown automaton $\mathcal{P}'$ such that*

$$\mathtt{PG}(\mathcal{P}') \quad \sim \quad \mathtt{PG}(\mathcal{P}) \parallel F$$

Sketch of the Proof: Let $\mathcal{P} = \langle Q, Act_{\mathcal{P}}, \Gamma, \delta, q_1, Z_1 \rangle$ and $F = \langle \mathcal{S}, Act_F, \rightarrow_F, s_0, \langle e_1, \ldots, e_z \rangle \rangle$. Then $\mathcal{P}'$ can be defined as $\langle Q \times \mathcal{S}, Act_{\mathcal{P}} \cup Act_F, \Gamma, \delta', \langle q_1, s_0 \rangle, Z_1 \rangle$ with:

- $\delta'(\langle q, s \rangle, a, Z) \ni \langle \langle q', s \rangle, \beta \rangle$ iff $\delta(q, a, Z) \ni \langle q', \beta \rangle$ and $a \notin Act_F$
- $\delta'(\langle q, s \rangle, a, Z) \ni \langle \langle q, s' \rangle, Z \rangle$ iff $s \xrightarrow{a}_F s'$ and $a \notin Act_{\mathcal{P}}$
- $\delta'(\langle q, s \rangle, a, Z) \ni \langle \langle q', s' \rangle, \beta \rangle$ iff $\delta(q, a, Z) \ni \langle q', \beta \rangle, s \xrightarrow{a}_F s'$ and $a \in Act_{\mathcal{P}} \cap Act_F$

A detailed proof of the theorem will be given in the full version of the paper.

2.2.1 Example

We illustrate the construction by means of the process management system given in Figure 3. Here the process handler `PH` can be interpreted as a system that allows to create processes via the action `create` and to terminate active processes via the action `term`. The process management system offers its service until it is 'shut-down' by the controller `C` via the action `shutdown`, which causes the process management system to deadlock after having terminated all active processes. Figure 5 illustrates the process graph that results from the parallel composition of the process handler `PH` and the controller `C`.

3 Specifying Behavioural Properties

The logic we consider is the alternation-free fragment of the modal mu-calculus [Koz83, Lar88]. For technical reasons, however, we represent formulas by means of *hierarchical (mutually recursive) equational systems* (cf. [CS91, CS92]), which are equally expressive as this fragment (cf. [CKS92]).

3.1 Syntax and Semantics of Basic Formulas

Let *Var* be a (countable) set of variables, and *Act* be a set of actions. In what follows, X will range over *Var*, and a over *Act*. Then the syntax of *basic* formulas is given by the following grammar.[4]

$$\Phi ::= tt \mid ff \mid X \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \langle a \rangle \Phi \mid [a]\Phi$$

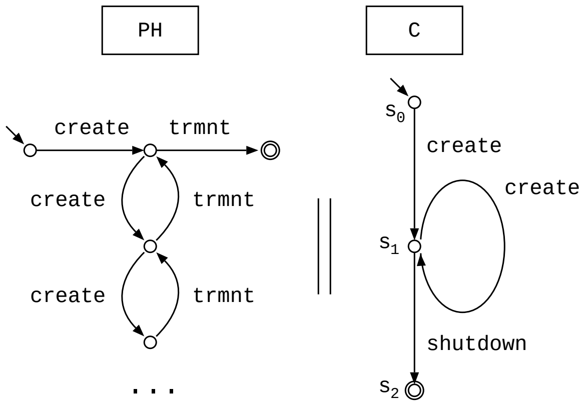

Figure 3: The Process Management System

$$\mathcal{PH} = \langle Q, Act, \{ Z_1 \}, \delta, q_1, Z_1 \rangle$$

where

- $Q = \{ q_1, q_2 \}$
- $Act = \{ \texttt{create}, \texttt{term} \}$ and
- δ is defined as follows:

$$\begin{array}{lcl} \delta(q_1, \texttt{create}, Z_1) & = & \{ \langle q_2, Z_1 \rangle \} \\ \delta(q_2, \texttt{create}, Z_1) & = & \{ \langle q_2, Z_1 Z_1 \rangle \} \\ \delta(q_2, \texttt{term}, Z_1) & = & \{ \langle q_2, \epsilon \rangle \} \end{array}$$

Figure 4: The Pushdown Automaton modelling the Process Handler.

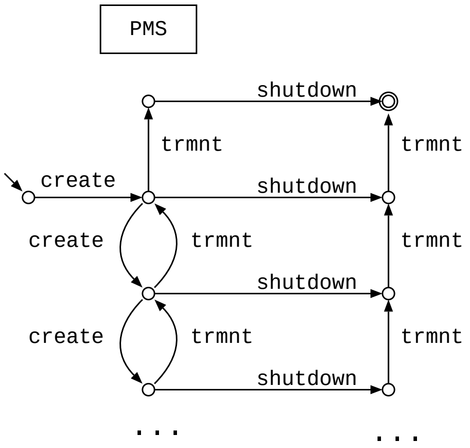

Figure 5: The Process Graph of the Process Management System

$$\mathcal{PMS} = \langle Q', Act', \{ Z_1 \}, \delta', q_1, Z_1 \rangle$$

where

- $Q' = Q \times \{ s_0, s_1, s_2 \}$
- $Act' = \{ \texttt{create}, \texttt{term}, \texttt{shutdown} \}$ and
- δ' is defined as follows:

$$\begin{array}{lcl} \delta'(\langle q_1, s_0 \rangle, \texttt{create}, Z_1) & = & \{ \langle \langle q_2, s_1 \rangle, Z_1 \rangle \} \\ \delta'(\langle q_1, s_1 \rangle, \texttt{create}, Z_1) & = & \{ \langle \langle q_2, s_1 \rangle, Z_1 \rangle \} \\ \delta'(\langle q_2, s_0 \rangle, \texttt{create}, Z_1) & = & \{ \langle \langle q_2, s_1 \rangle, Z_1 Z_1 \rangle \} \\ \delta'(\langle q_2, s_1 \rangle, \texttt{create}, Z_1) & = & \{ \langle \langle q_2, s_1 \rangle, Z_1 Z_1 \rangle \} \\ \delta'(\langle q_2, s_i \rangle, \texttt{term}, Z_1) & = & \{ \langle \langle q_2, s_i \rangle, \epsilon \rangle \}, \\ & & i = 0, 1, 2 \\ \delta'(\langle q_1, s_1 \rangle, \texttt{shutdown}, Z_1) & = & \{ \langle \langle q_1, s_2 \rangle, Z_1 \rangle \} \\ \delta'(\langle q_2, s_1 \rangle, \texttt{shutdown}, Z_1) & = & \{ \langle \langle q_2, s_2 \rangle, Z_1 \rangle \} \end{array}$$

Figure 6: The Pushdown Automaton modelling the Process Management System.

The formal semantics appears in Figure 7. It is given with respect to a labelled transition system $\langle \mathcal{S}, Act, \rightarrow \rangle$, and an environment ρ mapping variables to subsets of $\mathcal{S}$, where $\rho\,[\, X/S \,]$ is the environment resulting from ρ by updating the binding of X to S.

[4]In order to simplify our presentation, we only assume the atomic propositions *tt* and *ff* here. However, an extension covering arbitrary *atomic propositions*, which are consistent with the representation of the transition system under consideration, is straightforward.

$$
\begin{array}{rcl}
[\![tt]\!]\rho & = & \mathcal{S} \\
[\![ff]\!]\rho & = & \emptyset \\
[\![X]\!]\rho & = & \rho(X) \\
[\![\Phi_1 \vee \Phi_2]\!]\rho & = & [\![\Phi_1]\!]\rho \cup [\![\Phi_2]\!]\rho \\
[\![\Phi_1 \wedge \Phi_2]\!]\rho & = & [\![\Phi_1]\!]\rho \cap [\![\Phi_2]\!]\rho \\
[\![\langle a \rangle \Phi]\!]\rho & = & \{\, s \mid \exists\, s'.\, s \xrightarrow{a} s' \wedge s' \in [\![\Phi]\!]\rho \,\} \\
[\![[a]\Phi]\!]\rho & = & \{\, s \mid \forall\, s'.\, s \xrightarrow{a} s' \Rightarrow s' \in [\![\Phi]\!]\rho \,\}
\end{array}
$$

Figure 7: The Semantics of Basic Formulas.

Intuitively, the semantic function maps a formula to the set of states for which the formula is "true". Accordingly, all states satisfy tt, while no state satisfies ff. Furthermore s satisfies X if s is an element of the set bound to X in ρ. The propositional constructs are interpreted in the usual fashion: s satisfies $\Phi_1 \vee \Phi_2$ if it satisfies one of the Φ_i and $\Phi_1 \wedge \Phi_2$ if it satisfies both of them. The constructs $\langle a \rangle$ and $[a]$ are *modal operators*; s satisfies $\langle a \rangle \Phi$ if it has an a-derivative satisfying Φ, while s satisfies $[a]\Phi$ if each of its a-derivatives satisfy Φ.

3.2 Syntax of Hierarchical Equational Systems

By themselves, basic formulas do not provide much expressive power because they may only describe properties about a finite part of a system's behaviour. To remedy this lack of expressiveness, formulas may also be defined using *hierarchical equational systems*. These are equally expressive as the fragment of the modal mu-calculus which does not contain *alternating fixed points* [EL86].

Definition 3.1

- *An* (equational) block *has one of two forms,* $\texttt{min}\{Eq\}$ *or* $\texttt{max}\{Eq\}$*, where Eq is a list of (mutually recursive) equations* $\langle X_1 = \Phi_1, \ldots, X_n = \Phi_n \rangle$ *in which each Φ_i is a basic formula and the X_i are all distinct.*
- *An* equational system $E = \langle B_1, \ldots, B_m \rangle$ *is a list of equational blocks where the variables appearing on the left-hand sides of the blocks, denoted by $Var(E)$, are all distinct.*
- *An equational system $E = \langle B_1, \ldots, B_m \rangle$ is* hierarchical *if the existence of a left-hand-side variable of a block B_j in a right-hand-side formula of a block B_i implies $i \leq j$.*

3.3 Semantics of Hierarchical Equational Systems

To define the semantics of an hierarchical equational system E, we first define the semantics of an individual block. Let $Eq = \langle X_1 = \Phi_1, \ldots, X_n = \Phi_n \rangle$ be a list of equations. Then, given a fixed environment ρ, we may build a function $f^{\rho}_{Eq} : (2^{\mathcal{S}})^n \rightarrow (2^{\mathcal{S}})^n$ as follows. Let $\overline{S} = \langle S_1, \ldots, S_n \rangle \in (2^{\mathcal{S}})^n$, and let $\rho_{\overline{S}} = \rho[X_1 \mapsto S_1, \ldots, X_n \mapsto S_n]$ be the environment that results from ρ by updating the binding of X_i to S_i. Then $f^{\rho}_{Eq}(\overline{S}) = \langle [\![\Phi_1]\!]\rho_{\overline{S}}, \ldots, [\![\Phi_n]\!]\rho_{\overline{S}} \rangle$.

$(2^{\mathcal{S}})^n$ forms a complete lattice, where the ordering, join and meet operations are the pointwise extensions of the set-theoretic inclusion $\subseteq$, union $\cup$ and intersection $\cap$, respectively. Moreover, for any equation list Eq and environment ρ, f^{ρ}_{Eq} is monotonic with respect to this lattice and therefore, according to Tarski's fixed-point theorem [Tar55], has both a *greatest* fixed point, νf^{ρ}_{Eq}, and a *least* fixed point, μf^{ρ}_{Eq}. In general, these may be characterized as follows.

$$\nu f^{\rho}_{Eq} = \bigcup\{\, \overline{S} \mid \overline{S} \subseteq f^{\rho}_{Eq}(\overline{S})\,\}, \qquad \mu f^{\rho}_{Eq} = \bigcap\{\, \overline{S} \mid f^{\rho}_{Eq}(\overline{S}) \subseteq \overline{S}\,\}$$

Blocks $\mathtt{max}\{Eq\}$ and $\mathtt{min}\{Eq\}$ are now interpreted as the "greatest" fixed point of Eq, respectively the least, yielding *environments* in the following fashion.

$$[\![\mathtt{max}\{Eq\}]\!]\rho = \rho_{\nu f^{\rho}_{Eq}}, \qquad [\![\mathtt{min}\{Eq\}]\!]\rho = \rho_{\mu f^{\rho}_{Eq}}$$

Given ρ, we finally define the relative semantics of an hierarchical equational system $E = \langle B_1, \ldots, B_m\rangle$ in terms of the following sequence of environments: $\rho_m = [\![B_m]\!]\rho, \ldots, \rho_1 = [\![B_1]\!]\rho_2$ as $[\![E]\!]\rho = \rho_1$.

The definition of what it means for a state in a labelled transition system to satisfy a basic formula Φ whose variables are "bound" by an hierarchical equational system E is now straightforward. First, we say that Φ is *closed* with respect to E if every variable in Φ is a member of $Var(E)$. We also refer to E as *closed* if each right-hand side in each block in E is closed with respect to E. Henceforth, the set of all closed hierarchical equational systems is denoted with $\mathcal{E}$.

It turns out that it is sufficient to deal with the case where Φ is a variable and that $[\![E]\!]\rho$ does not depend on ρ for closed equational systems E (cf. [BS92]). Thus when E is closed, we omit reference to ρ. Furthermore, we abbreviate $s \in [\![X_1]\!]_{[\![E]\!]}$ with $s \models E$, when X_1 is the first variable in the first block of E. Finally, given a process graph $G = \langle \mathcal{S}, Act, \rightarrow, s_0, \{\, e_1, \ldots, e_z \,\}\rangle$ we write $G \models E$ if $s_0 \models E$. The generalization for a set of hierarchical equational systems $M \subseteq \mathcal{E}$ is written as $G \models M$, meaning $G \models E$ for all $E \in M$.

4 Verifying Behavioural Properties

Our algorithm is based on a 'tree-like' decomposition of pushdown automata and an extension of the higher order model checker presented in [BS92], which allows us to deal with the finite-state control of a pushdown automaton. The heart of the algorithm consists of determining *property transformers* for each part of the pushdown automaton decomposition under consideration, which provide a compositional and sound description of the logical behaviour of the pushdown automaton.

In Section 4.1 we present the decomposition of a pushdown automaton, which allows us to characterize the underlying transition graph in a modular way. That using property transformers provides a sound basis for model-checking follows from the higher-order formulation of the semantics of process graphs in terms of its valid formulas (cf. Section 4.2). Finally, we present our model-checking algorithm. The effectiveness of our method is a consequence of the fact that it suffices to deal with the part of a property transformer which concerns the subformulas of the property to be investigated (cf. Section 4.3).

4.1 Expansion and Decomposition

Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a pushdown automaton. Then we denote with $\mathcal{P}^{(q,\gamma)}, q \in Q, \gamma \in \Gamma^*$ the "intermediate" pushdown automaton $\mathcal{P}' = \langle Q, Act, \Gamma, \delta, q, \gamma \rangle$ which starts with state q and stack contents γ. Process graphs of the form $[q, \gamma] =_{\rm df} \mathtt{PG}(\mathcal{P}^{(q,\gamma)})$ are called *fragments* of the pushdown automaton $\mathcal{P}$. Of specific interest are fragments of the form $[q, Z]$ which we call *simple*.

Fragments of a given pushdown automaton all have the same arity z: there is an end state for each of the states of the finite control. This similarity in structure can be exploited in order to identify the end states of different process graphs that correspond to the same state of the finite control. Thus, in order to guarantee a concise representation, we use the following derived notions of the usual sequential composition

$$G_0 \;;' \langle G_1, \ldots, G_z \rangle) \;=_{\rm df}\; (G_0 \;;\; \langle G_1, \ldots, G_z \rangle)\,[e_k^{(i)} = e_k^{(j)}, i, j, k \in \{\,1, \ldots, z\,\}]$$

and nondeterministic choice

$$G_1 +' G_2 \;=_{\rm df}\; (G_1 + G_2)\,[e_i^{(1)} = e_i^{(2)}, i \in \{\,1, \ldots, z\,\}]$$

The following decomposition result is central:

Theorem 4.1 *Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a pushdown automaton $q \in Q = \{q_1, \ldots, q_n\}$, and $Z\gamma \in \Gamma^+$. Then we have*

1. **Expansion Law:** $\quad [q, Z\gamma] \;\sim\; +' \{ \stackrel{a}{\rightarrow} ;' [q', \beta\gamma] \;\mid\; \delta(q, a, Z) \ni \langle q', \beta \rangle \}$
   where $\stackrel{a}{\rightarrow}$ denotes the process graph which consists of the single transition $s_0 \stackrel{a}{\rightarrow} s_1$.

2. **Decomposition Law:** $\quad [q, Z\gamma] \;\sim\; [q, Z] \;;' \langle [q_1, \gamma], \ldots, [q_n, \gamma] \rangle$

Fact 1 follows directly from the definition of a process graph of a pushdown automaton, while Fact 2 formalizes the following intuition: a pushdown automaton with stack contents $Z\gamma$ which starts in state q and reaches state q' after deleting Z from the stack, continues with state q' and stack contents γ.

That two processes which are equivalent up to bisimulation satisfy the same set of μ-formulas (cf. [HM85]) guarantees that we can use Theorem 4.1 to finitely decompose the pushdown automaton under consideration without changing the set of valid formulas.

4.2 Higher Order Semantics

The validity of a formula is defined with respect to single states. This notion cannot directly be adapted to fragments: the truth value at the start state might not be the same in different contexts, i.e. for different continuations[5]. However, if two lists of continuations satisfy the same sets of formulas, then so does the start state of the fragment in these both contexts. This observation is the key to the *higher order semantics* of [BS92]: It is consistent with usual semantics to view a process graph as a *property transformer*, i.e. as a function which yields the set of formulas valid at the start state, relative to the assumption that the tuple of sets of formulas to which the transformer is applied are valid at the end states.

[5]Note that we are dealing with forward-modalities. Thus the validity of formulas "propagates" backward.

Definition 4.2 (Property Transformer)
Let G be a process graph of arity z. Then we interpret G as the function $[\![G]\!] : (2^{\mathcal{E}})^z \to 2^{\mathcal{E}}$, called the property transformer *of G, which is defined as follows.*

$$[\![G]\!](M_1, \ldots, M_z) = \{ \; E \in \mathcal{E} \mid \forall \; G_1, \ldots, G_z \; \in \mathcal{G} \\ (\forall \; 1 \leq i \leq z \quad G_i \; \models \; M_i \,) \implies G \; ; \; \langle G_1, \ldots, G_z \rangle \; \models \; E \}$$

That the higher order semantics is consistent with the usual semantics of process graphs in terms of its valid formulas, is a consequence of the following theorem.

Theorem 4.3 (Consistency of the Higher Order Semantics)
Let G be a process graph and E be a closed equational system. Then we have:

$$G \models E \qquad \Longleftrightarrow \qquad E \in [\![G]\!](M_\emptyset, \ldots, M_\emptyset),$$

where $M_\emptyset \subseteq \mathcal{E}$ is the set of all propositions that are "true" at an end state , i.e.:

$$M_\emptyset =_{\mathrm{df}} \{ \, E \mid e \models E \quad \textit{in the labelled transition system } \langle \{ \, e \, \}, Act, \emptyset \rangle \, \}$$

The point of the higher order semantics is its *compositionality* with respect to sequential composition:

Theorem 4.4 (Compositionality of Higher Order Semantics)
Let $G, G_1, \ldots, G_z$ be process graphs, where G has arity z. Then we have:

$$[\![G \; ; \; \langle G_1, \ldots, G_z \rangle]\!] = [\![G]\!] \;\circ\; \langle [\![G_1]\!], \ldots, [\![G_z]\!] \rangle$$

The extension of the higher order semantics for process graphs to simple fragments of pushdown automata is straightforward. Associating with each fragment the property transformer of the underlying process graph leads to the following general inductive definition.

Definition 4.5 *Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a pushdown automaton. Then we define:*

$$\begin{array}{lll} [\![q_i, \epsilon]\!] & =_{\mathrm{df}} & \mathtt{proj}_i \\ [\![q, Z]\!] =_{\mathrm{df}} [\![\mathtt{PG}(\mathcal{P}^{(q,Z)})]\!] & & \\ [\![q_i, Z\gamma]\!] & =_{\mathrm{df}} & [\![q_i, Z]\!] \;\circ\; \langle [\![q_1, \gamma]\!], \ldots, [\![q_n, \gamma]\!] \rangle \end{array}$$

where $\mathtt{proj}_i$ means the projection on the i-th argument. [6]

As a consequence of the decomposition theorem and the compositionality of the higher order semantics we obtain:

Proposition 4.6 *Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a pushdown automaton. Then we have:*

$$[\![q, \gamma]\!] =_{\mathrm{df}} [\![\mathtt{PG}(\mathcal{P}^{(q,\gamma)})]\!]$$

[6] Note that in the fragment $[q, \epsilon]$, which consists of an infinite number of states, only one end state, namely $\langle q_i, \epsilon \rangle$, is reachable. The associated property transformer is thus the projection on the i-th argument.

4.3 The Model-Checking Algorithm

In this section, we generalize the model-checking algorithm of [BS92] in order to deal with *pushdown* processes. It consists of three steps:

1. Construction of a *recursive function scheme* E_{PT} with respect to the pushdown automaton and the hierarchical equation system E of interest: The appropriate solution of this scheme, called the *PT-equational system*, will yield the property transformers of each fragment of the pushdown automaton restricted to the subformulas of E. Appendix A gives the formal details of the construction for E_{PT}, which obviously does not contain more than $|E| * |Q| * |\delta| * pl(\delta)$ equations.

2. Solving of the PT-equational system E_{PT} by component-wise computation of a *property transformer* for each fragment of the pushdown automaton: essentially, we proceed as in the finite-state case (cf. [CKS92]), except for

   - the domain for the iteration, which is higher order here, i.e. it consists of property transformers.
   - the handling of fragments. They are dealt with by applying the currently valid approximation of their property transformers.

   The computation of the *component property transformers* (CPT) consists of two phases: the initialization and the fixpoint computation.

   The initialization of the CPTs depends on whether we process a *min* or a *max*-block: a CPT is initialized with the maximal transformer in case X is a *max*-variable and with the minimal transformer otherwise. This reflects the fact that *max*-variables are initially assumed to be satisfied by all states and *min*-variables by none.

   The overall fixpoint computation is done by standard fixpoint iteration. The evaluation of the right-hand sides during the fixpoint computation simply consists of functional composition and the application of the meet and join operations of the lattice of all CPT's (cf. [BS92]). Details of the CPT computation can be found in Appendix B.

3. Decision of the model-checking problem: Having the predicate transformers at hand, the model-checking problem can be solved by checking whether the considered formula is a member of the set of formulas that results from applying the property transformer associated with the initial fragment of the "main procedure" to the tuple of sets of formulas that are valid for a terminated (deadlocked) state.

4.3.1 Correctness and Complexity

The procedure **solve** consists of an initialization, a call of **update** and a computation of $PT^{X_1}_{[q_1,Z_1]}(M_\emptyset, \ldots, M_\emptyset)$. It always terminates, since the number of simple fragments is finite and the procedure **update** is monotonic on the finite lattice $(2^{\mathcal{X}})^{|Q|} \rightarrow 2^{\mathcal{X}}$. Moreover, upon termination of **update**, a single application of $PT^{X_1}_{[q_1,Z_1]}$ suffices to decide whether the formula represented by X_1 is valid for the overall system, i.e. at the start state of the process graph of the pushdown automaton. This is a consequence of the following stronger property.

Theorem 4.7 (Correctness)
Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a pushdown automaton and E be a closed hierarchical equational system. Then, upon termination of the algorithm **solve**, *we have*

$$\forall q \in Q, Z \in \Gamma. \quad PT_{[q,Z]} \;=\; [\![q, Z]\!]_{|\,E},$$

where $[\![q, Z]\!]_{|\,E}$ is the restriction of the property transformer $[\![q, Z]\!]$ to the variables of E.

In particular, the theorem yields

$$PT_{[q_1,Z_1]}(M_\emptyset, \ldots, M_\emptyset) \;=\; [\![q_1, Z_1]\!]_{|\,E}(M_\emptyset, \ldots, M_\emptyset)$$

Thus the algorithm computes the set of all variables of E which are valid at the start state of the process graph defined by $\mathcal{P}$.

Let us now consider the complexity of our algorithm. Here it turns out that the maximal length of all strings the transition relation δ can push onto the stack and the *weight* of a property are important: Given a pushdown automaton $\mathcal{P}$ with transition relation δ, we define the *push length* of δ as $pl(\delta) = max\{\, |\beta| \mid \exists\, q, q' \in Q, a \in Act, Z \in \Gamma.\ \langle q', \beta \rangle \in \delta(q, a, Z)\,\}$. Moreover, for a given equational system E, we define the *weight of E* as the number of distinct variables that occur after a modality on the right-hand side in some equation of E. Denoting the weight of E by $w(E)$, we have:

Theorem 4.8 (Complexity)
Let $\mathcal{P}$ be a pushdown automaton and E be a closed hierarchical equational system. Then the worst-case time complexity of **solve** *is*

$$O(\ |E| * |Q|^2 * |\Gamma| * |\delta| * (w(E)^2 * |Q| * pl(\delta) + |E| * |\Gamma|) * 4^{w(E)*|Q|}\ )$$

Sketch of the Proof: Let us first consider the time complexity for computing a new approximate value for a right-hand side of a decomposition equation (cf. Appendix A). In such a PT-equation the right-hand side can be evaluated by computing the new CPT argument-wise for each of the $(2^{|E|})^{|Q|}$ arguments. The neccessary functional composition can then be computed in $O(\ |E| * |Q|\ )$. A hierarchical computation of the at most $|E| * |Q| * |\delta| * pl(\delta)$ decomposition equations gives a time complexity of $O(\ |E|^2 * |Q|^2 * |\delta| * pl(\delta) * 2^{|E|*|Q|}\ )$. However, as a closer analysis of the algorithm reveals, only CPT's for variables which counts as a weight are of interest. Thus the factor $|E|$ can be reduced to $w(E)$.

The next step of the algorithm computes a new approximation for the property transformer choosen from the workset. The most expensive right-hand side computations result from PT-equations associated with an equation $X = \Phi$ where Φ contains a modality. Moreover, the right-hand side of $PT^X_{[q,Z]}$ consists of an k-ary meet or join, where k is the number of transitions $\langle q', \beta \rangle \in \delta(q, a, Z)$ which is bound by $|\delta|$. Thus the new approximate value can be computed in $O(\ |\delta| * 2^{w(E)*|Q|}\ )$.

Since after $|E| * |Q| * |\Gamma|$ computations at least one CPT must have a better approximation (unless we have reached the fixed-point) and the maximal chain length arising during the fixpoint computation of each of the $|E| * |Q| * |\Gamma|$ CPTs is less than $2^{w(E)*|Q|}$, the overall worst-case time complexity for computing the fixpoint can be estimated by

$$O(\ |E| * |Q| * |\Gamma| * 2^{w(E)*|Q|} * (w(E)^2 * |Q|^2 * |\delta| * pl(\delta) * 2^{w(E)*|Q|} + |E| * |Q| * |\Gamma| * |\delta| * 2^{w(E)*|Q|})\ ).$$

□

5 Conclusions and Future Work

We have shown that the class of pushdown processes, which strictly generalizes the class of context-free processes, is 1) closed under parallel composition with finite state systems, and can 2) be model checked by means of an elegant adaptation to pushdown automata of the higher order model checker introduced in [BS92]. In fact, pushdown processes turn out to be the appropriate generalization of context-free processes for the automatic verification of distributed systems: any nontrivial parallel composition involving context-free processes requires at least the more general structure of pushdown processes. Moreover, allowing parallel compositions of two context-free processes yields the full power of Turing machines, which excludes automatic verification.

In [Ste91, Ste93] a method has been presented that allows us to generate data flow analysis algorithms from modal specifications by partially evaluating a model checker with respect to the specifying formula. As a major application of the extended model checking algorithm presented in this paper we are currently generalizing this method to the interprocedural case.

Acknowledgements

We would like to thank Hardi Hungar for his constructive comments.

References

[Bra91] J.C. Bradfield. *Verifying Temporal Properties of Systems with Applications to Petri Nets*. PhD thesis, University of Edinburgh, 1991.

[BS91] J.C. Bradfield and C. Stirling. Local Model Checking for Infinite State Spaces. Technical Report ECS-LFCS-90-115, LFCS, Jun 1991.

[BS92] O. Burkart and B. Steffen. Model Checking for Context-Free Processes. In *CONCUR '92, LNCS 630*, pages 123–137. Springer, 1992.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CKS92] R. Cleaveland, M. Klein, and B. Steffen. Faster Model Checking for the Modal Mu-Calculus. In *CAV '92, LNCS 663*, pages 410–422, 1992.

[Cle90] R. Cleaveland. Tableau-Based Model Checking in the Propositional Mu-Calculus. *Acta Informatica*, 27:725–747, 1990.

[CM90] D. Caucal and R. Monfort. On the Transition Graphs of Automata and Grammars. In *Graph-Theoretic Concepts in Computer Science, LNCS 484*, pages 311–337. Springer, 1990.

[CS91] R. Cleaveland and B. Steffen. Computing Behavioural Relations, Logically. In *ICALP '91, LNCS 510*, pages 127–138. Springer, 1991.

[CS92] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. In *CAV '91, LNCS 575*, pages 48–58. Springer, 1992.

[EL86] E.A. Emerson and C.-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proc. 1th Annual Symp. on Logic in Computer Science*, pages 267–278. IEEE Computer Society Press, 1986.

[HM85] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, 32(1):137–161, jan 1985.

[Hoa88] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall, 1988.

[HS93] H. Hungar and B. Steffen. Local Model-Checking for Context-Free Processes. In *ICALP '93, LNCS 700*, pages 593–605, 1993.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Lar88] K.G. Larsen. Proof Systems for Hennessy–Milner Logic with Recursion. In *CAAP '88, LNCS 299*, pages 215–230. Springer, 1988.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice–Hall, 1989.

[MS85] D.E. Muller and P.E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theoretical Computer Science*, 37:51–75, 1985.

[Ste91] B. Steffen. Data Flow Analysis as Model Checking. In *TACS '91, LNCS 526*, pages 346–365. Springer, 1991.

[Ste93] B. Steffen. Generating Data Flow Analysis Algorithms from Modal Specifications. To appear in *Science of Computer Programming*, 1993.

[SW89] C. Stirling and D. Walker. Local Model Checking in the Modal Mu-Calculus. In *TAPSOFT '89, LNCS 351*, pages 369–383. Springer, 1989.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

A Constructing the PT-Equational System

For simplicity, we present only the construction for a single minimal equational block here, which can straightforwardly be extended to arbitrary hierarchical equational systems.

Let $\mathcal{P} = \langle Q, Act, \Gamma, \delta, q_1, Z_1 \rangle$ be a PDA with $|Q| = n$ and $pl(\delta) =_{\mathrm{df}} max\{ |\beta| \mid \exists\, q, q' \in Q, a \in Act, Z \in \Gamma.\ \langle q', \beta \rangle \in \delta(q, a, Z) \}$. Moreover, let $B = \mathtt{min}\{\langle X_1 = \Phi_1, \ldots, X_m = \Phi_m \rangle\}$ be a closed minimal equational block with left-hand side variables $\mathcal{X} = \{ X_i \mid 1 \leq i \leq m \}$, where X_1 represents the property to be investigated. In order to simplify the algorithm, we split each property transformer $PT_{[q,\gamma]} : (2^{\mathcal{X}})^n \rightarrow 2^{\mathcal{X}}$ ($q \in Q, \gamma \in \Gamma^*$ with $|\gamma| \leq pl(\delta)$) into its *component property transformers* (CPT) $PT^X_{[q,\gamma]} : (2^{\mathcal{X}})^n \rightarrow \mathbb{B}$. Given $X \in \mathcal{X}$, $PT^X_{[q,\gamma]}$ is defined by $PT^X_{[q,\gamma]} =_{\mathrm{df}} \chi^X \circ PT_{[q,\gamma]}$, where χ^X denotes the predicate characterizing the membership of X and $\mathbb{B}$ is the Boolean lattice over $\{0,1\}$. Note that the CPTs $PT^X_{[q,\gamma]}$ for a given $[q,\gamma]$ are equivalent to $PT_{[q,\gamma]}$, as $PT_{[q,\gamma]} = \bigcup_{X \in \mathcal{X}} \delta_X \circ PT^X_{[q,\gamma]}$, where $\delta_X(0) = \emptyset$ and $\delta_X(1) = \{X\}$.

Let now $\perp^{(n)}$ denote the least CPT which maps every element of $(2^{\mathcal{X}})^n$ to 0, $\top^{(n)}$ the greatest CPT which maps every element of $(2^{\mathcal{X}})^n$ to 1 and $\chi_n^{X \in j} : (2^{\mathcal{X}})^n \rightarrow \mathbb{B}$ be the predicate characterizing whether X is a member of the j-th argument.

Then the PT-equational system E_{PT} is built by means of the rules specified in Figure 8. The obtained PT-equational system consists of two parts: a set of PT-equations which are related to the property represented by the equational block B and a set of PT-equations defining the property transformers of non-simple fragments using the Decomposition Law. Note that given an approximation for the property transformers of all simple fragments the approximation for the property transformers of the non-simple fragments can be computed hierarchically by evaluating every right-hand-side only once.

Property related equations

$PT^X_{[q,Z]}$	$=$	$\top^{(n)}$	$\in E_{PT}$,	if	X	$=$	tt	$\in B$
$PT^X_{[q,Z]}$	$=$	$\perp^{(n)}$	$\in E_{PT}$,	if	X	$=$	ff	$\in B$
$PT^X_{[q,Z]}$	$=$	$PT^{X'}_{[q,Z]}$	$\in E_{PT}$,	if	X	$=$	X'	$\in B$
$PT^X_{[q,Z]}$	$=$	$PT^{X'}_{[q,Z]} \sqcup PT^{X''}_{[q,Z]}$	$\in E_{PT}$,	if	X	$=$	$X' \vee X''$	$\in B$
$PT^X_{[q,Z]}$	$=$	$PT^{X'}_{[q,Z]} \sqcap PT^{X''}_{[q,Z]}$	$\in E_{PT}$,	if	X	$=$	$X' \wedge X''$	$\in B$
$PT^X_{[q,Z]}$	$=$	$\bigsqcup_{\langle q',\beta \rangle \in \delta(q,a,Z)} PT^{X'}_{[q',\beta]}$	$\in E_{PT}$,	if	X	$=$	$\langle a \rangle X'$	$\in B$
$PT^X_{[q,Z]}$	$=$	$\bigsqcap_{\langle q',\beta \rangle \in \delta(q,a,Z)} PT^{X'}_{[q',\beta]}$	$\in E_{PT}$,	if	X	$=$	$[a] X'$	$\in B$

Decomposition equations

$PT^X_{[q_i,\epsilon]}$	$=$	$\chi_n^{X \in i}$	$\in E_{PT}$	
$PT^X_{[q_i,Z\gamma]}$	$=$	$PT^X_{[q_i,Z]} \circ \langle PT_{[q_1,\gamma]}, \ldots, PT_{[q_n,\gamma]} \rangle$	$\in E_{PT}$,	if $PT^X_{[q_i,Z\gamma]}$ occurs in some right-hand-side of E_{PT}

Figure 8: The rules for constructing E_{PT}

B The model-checking algorithm

The following initialization procedure simply distinguishes *min*- and *max*-variables:

procedure init
`BEGIN`

$$\forall q \in Q, Z \in \Gamma, X \in \mathcal{X}. \qquad PT^X_{[q,Z]} := \begin{cases} \bot^{(n)}, & \text{if X is a } \mathit{min}\text{-variable} \\ \top^{(n)}, & \text{if X is a } \mathit{max}\text{-variable} \end{cases}$$

`END`

The central procedure **compfix**, which works along the lines of [BS92], updates the component property transformers $PT^X_{[q,Z]}$ until consistency is reached. Denoting the right-hand side term of $PT^X_{[q,Z]}$ in E_{PT} by $rhs(PT^X_{[q,Z]})$, **compfix** is given by:

procedure compfix
`BEGIN`
$\quad workset := \{\, PT^X_{[q,Z]} \mid q \in Q, Z \in \Gamma, X \in \mathcal{X} \,\}$
$\quad$ `WHILE` $workset \neq \emptyset$ `DO`
$\quad\quad$ Compute hierarchically $PT^X_{[q,\gamma]}$ for all Decomposition Equations.
$\quad\quad$ `LET` $PT^X_{[q,Z]} \in workset$
$\quad\quad$ `BEGIN`
$\quad\quad\quad workset := workset \setminus \{PT^X_{[q,Z]}\}$
$\quad\quad\quad PT.old := PT^X_{[q,Z]}$
$\quad\quad\quad PT^X_{[q,Z]} := \texttt{eval}(rhs(PT^X_{[q,Z]}))$
$\quad\quad\quad$ `IF` $PT^X_{[q,Z]} \neq PT.old$ `THEN`
$\quad\quad\quad\quad workset := workset \cup \{\, PT^{X'}_{[q',Z']} \mid PT^X_{[q,Z]} \in rhs(PT^{X'}_{[q',Z']}) \,\}$
$\quad\quad\quad$ `FI`
$\quad\quad$ `END`
$\quad$ `OD`
`END`

Now our overall model-checking algorithm can be summarized as follows:

procedure solve$(E, \mathcal{P})$
`BEGIN`
$\quad$ Construction of the PT-equational system E_{PT}.
$\quad$ **init**
$\quad$ **compfix**
$\quad$ `RETURN` $PT^{X_1}_{[q_1,Z_1]}(M_\emptyset, \ldots, M_\emptyset)$
`END`