[Bry] Bryant, R.E. "Graph-Based Algorithms for Boolean Function Manipulation" *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677-691, 1986

[CES] Clarke, E.M., 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*, v. 8, n. 2, 1986, pp. 244–263.

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

[CPS] Cleaveland, R., J. Parrow and B. Steffen. "The Concurrency Workbench." In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, 1989, Lecture Notes in Computer Science 407, pp. 24–37. Springer-Verlag, Berlin.

[CS1] Cleaveland, R. and B. Steffen. "A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus." In *Proceedings CAV '91*, 1991

[CS2] Cleaveland, R. and B. Steffen. "Computing Behavioural Relations, Logically." In *Proceedings ICALP '91*, 1991.

[EL] Emerson, E.A. and C.-L. Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus." In *Proceedings of the First Annual Symposium on Logic in Computer Science*, 1986, pp. 267–278.

[HS] Hüttel, H. and Stirling, C. "Actions Speak Louder than Words." In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science*, 1991, pp. 376–386.

[Ko] Kozen, D. "Results on the Propositional μ-Calculus." *Theoretical Computer Science*, v. 27, 1983, pp. 333–354.

[La] Larsen, K.G. "Proof Systems for Hennessy-Milner Logic with Recursion." In *Proceedings of CAAP*, 1988.

[SW] Stirling, C. and D. Walker. "Local Model Checking in the Modal Mu-Calculus." In *Proceedings of TAPSOFT '89*, Lecture Notes in Computer Science 351. Springer-Verlag, Berlin, 1989.

[Ta] Tarski, A. "A Lattice-Theoretical Fixpoint Theorem and its Applications." *Pacific Journal of Mathematics*, v. 5, 1955.

[Wi] Winskel, G. "Model Checking in the Modal ν-Calculus." In *Proceedings of Eleventh International Colloquium on Automata, Languages and Programming*, 1989.

$$
max \left\{ \begin{array}{llll}
X_1 = X_2 \wedge X_3 & X_6 = [\#]X_1 & Y_1 = Y_2 \wedge Y_3 & Y_6 \;= [\#]Y_7 \\
X_2 = [a]X_1 & X_7 = X_8 \wedge X_9 & Y_2 = [a]Y_1 & Y_7 \;= \mathtt{ff} \\
X_3 = X_4 \wedge X_5 & X_8 = [a']Y_1 & Y_3 = Y_4 \wedge Y_5 & Y_8 \;= Y_9 \wedge Y_{10} \\
X_4 = [b]X_1 & X_9 = [b']Y_1 & Y_4 = [b]Y_1 & Y_9 \;= [a']Y_1 \\
X_5 = X_6 \wedge X_7 & & Y_5 = Y_6 \wedge Y_8 & Y_{10} = [b']Y_1
\end{array} \right\}
$$

5 Conclusions and Future Work

We have presented an algorithm that decides the alternation-free modal mu-calculus for *context-free* processes, i.e. for processes resulting from an extension of standard finite-state labeled transition systems by introducing *recursive procedures* or, alternatively, a *recursive action refinement mechanism.* The point of this algorithm is its *higher orderedness*: it iteratively determines a property transformer for the procedural components of the (infinite-state) system being analyzed. This idea leads to a model checking algorithm for infinite state systems, which is linear in the size of the system's representation and exponential in the size of the property.

The complexity result about our algorithm is encouraging, because in practice the process systems are extremely large, however, the weights of the properties to be investigated are often very small. For example for deadlock checking the weight would be one. Thus the exponential factor is often uncritical. The practibility of our approach may also increase significantly by representing the property transformer in terms of BDD's (cf. [Bry]). We are going to implement this algorithm in the Edinburgh Concurrency Workbench (cf. [CPS]), which currently can only deal with finite state systems, in order to validate this statement from the practical point of view. Moreover, we are working on an extension of the algorithm that covers the full modal mu-calculus.

Acknowledgements

We would like to thank Bengt Johnsson for his constructive comments.

References

[BBK] Baeten, J.C.M., J.A. Bergstra and J.W. Klop. "Decidability of bisimulation equivalence for processes generating context-free languages." In *LNCS 259*, 1987, pp. 93-114. Springer-Verlag, Berlin.

[BK] Baeten, J.C.M., and J.W. Klop "Process theory based on bisimulation semantics" In *LNCS 354*, 1988, pp. 50-122. Springer-Verlag, Berlin.

[Bra] Bradfield, J.C. "Verifying Temporal Properties of Systems with Applications to Petri Nets." *Ph.D. Thesis*, University of Edinburgh, 1991

In particular, the theorem yields

$$PT_{\texttt{start}}(\mathbf{deadlock}_{|\mathcal{B}}) = [\![\texttt{start}]\!]^{\tilde{\mathcal{T}}}_{|\mathcal{B}}(\mathbf{deadlock}_{|\mathcal{B}})$$

Thus the algorithm computes the set of variables of $\mathcal{B}$ which are "true" at the start state in the fully expanded transition system of P_{main}.

The proofs of the two inclusion that are necessary to establish Theorem 4.3 can be sketched as follows:

1. $PT_\sigma \sqsubseteq [\![\sigma]\!]^{\tilde{\mathcal{T}}}_{|\mathcal{B}}$ follows by straightforward induction on the number of iterations during the execution of the update procedure.

2. $PT_\sigma \sqsupseteq [\![\sigma]\!]^{\tilde{\mathcal{T}}}_{|\mathcal{B}}$ follows by induction on the depth of the "proof tree" which is necessary to establish the validity of a certain formula at a given state class.

Let us now consider the complexity of our algorithm. Here it turns out that the *weight* of a property is essential. Given a closed set of blocks $\mathcal{B}$, we define the weight of $\mathcal{B}$ denoted by $w(\mathcal{B})$ as the number of distinct variables that occur after a modality on the right-hand-side in some equation of $\mathcal{B}$. We have:

Theorem 4.4 (Complexity)
Let $\tilde{\mathcal{T}} = \{P_i\}_{i\in I}$ be a system of procedural transition systems and $\mathcal{B}$ be a closed set of blocks. Then the worst-case time complexity of **solve** *is* $O(|\Sigma| * |w(\mathcal{B})| * 2^{|w(\mathcal{B})|})$.

In order to prove this theorem, we note the following. The initialization takes time proportional to $|\Sigma|$. Furthermore, the complexity of the updating procedure is proportional to the product of the number of state classes and the maximal number of updating steps, which are necessary to compute a single property transformer. Because of the monotony of the updating procedure and the fact that we can restrict our attention to the variables that count as a weight for the property under investigation the maximal number of updating steps can be estimated by $|w(\mathcal{B})| * 2^{|w(\mathcal{B})|}$.

Continuation of the Example

Consider the encoding machine of Figure 1, and suppose that we are interested in proving that this machine reads the full (finite) sequence of input symbols, before it outputs the first code. As long as we restrict ourselves to finite input sequences terminated by #, this property can be reformulated by saying that after reading an encoded character a' or b' the machine will always refuse to read #. The following figure shows this property expressed as a *max*-block of equations, where X_1 represents the specified property.

Note that, although this *max*-block is rather large, its weight is just 3, because only the variables X_1, Y_1 and Y_7 occur after a modality on the right-hand-side in some equation of $\mathcal{B}$. The full version of the paper will discuss the application of our algorithm to this example in detail.

Note that this definition is well-founded because of the syntactic restriction that the graph induced by $\rightarrow$ must be acyclic. The procedure **solve** proceeds now as follows.

- Initialize each PT_σ for $\sigma \in \Sigma \setminus \{\, \mathtt{end}_{P_i} \mid i \in I \,\}$ with $PT_\emptyset$ and each $PT_{\mathtt{end}_{P_i}}$ for $i \in I$ with PT_{Id} respectively.

- Update the functions PT_σ according to the following procedure **update**.

$workset := \{\, PT_\sigma \mid \sigma \in \Sigma \,\}$
WHILE $workset \neq \emptyset$ DO
  LET $PT_\sigma \in workset$
    BEGIN
    $workset := workset \setminus \{PT_\sigma\}$
    $PT_\sigma.old := PT_\sigma$
    LET $\{\, \sigma \xrightarrow{\alpha_j} \sigma_j \mid j \in \{1, \ldots, n\} \,\}$ the set of all available transformations for σ
      $PT_\sigma := \Diamond^{\sigma}_{j=1,\ldots,n}\ PT_{[\![\alpha_j]\!]} \circ PT_{\sigma_j}$
    IF $PT_\sigma \neq PT_\sigma.old$ THEN
      IF $\sigma = \mathtt{start}_{P_i}$ for some $i \in I$ THEN
        $workset := workset \cup \{\, PT_{\sigma'} \mid \sigma' \xrightarrow{P_i}, \sigma' \in \Sigma \,\}$
      ELSE
        $workset := workset \cup \{\, PT_{\sigma'} \mid \sigma' \xrightarrow{\alpha} \sigma \,\}$
      FI
    FI
    END
OD

- Check whether $X_1 \in PT_{\mathtt{start}}(\mathbf{deadlock}_{|\mathcal{B}})$, where $\mathbf{deadlock}_{|\mathcal{B}}$ is the set of all variables of $\mathcal{B}$ valid at a "deadlock" state. It is defined as follows.

$$\mathbf{deadlock}_{|\mathcal{B}} \stackrel{\text{def}}{=} \{\, X_i \in X \mid s \models X_i \text{ where } \mathcal{B} \text{ in } \mathcal{T} = \langle \{\, s \,\}, Act, \emptyset \rangle \,\}.$$

4.4 Correctness and Complexity

The procedure **solve** consists of an initialization, a call of **update** and a computation of $PT_{\mathtt{start}}(\mathbf{deadlock}_{|\mathcal{B}})$. It always terminates, since the number of state classes is finite and the procedure **update** is monotonic on the finite lattice $\mathcal{D} \stackrel{\text{def}}{=} 2^X \rightarrow 2^X$. Moreover, upon termination of **update**, the application of $PT_{\mathtt{start}}$ yields the set of formulas that are valid at the start state of the "main procedure".

Theorem 4.3 (Correctness)
Let $\tilde{\mathcal{T}} = \{P_i\}_{i \in I}$ be a system of procedural transition systems and $\mathcal{B}$ be a closed set of equational blocks. Then we have upon termination of the algorithm

$$\forall \sigma \in \Sigma.\ PT_\sigma \ = \ [\![\sigma]\!]^{\tilde{\mathcal{T}}}_{|\mathcal{B}}$$

where $[\![\sigma]\!]^{\tilde{\mathcal{T}}}_{|\mathcal{B}}$ is the restriction of the property transformer $[\![\sigma]\!]^{\tilde{\mathcal{T}}}$ to the variables of $\mathcal{B}$.

This does not impose any restriction, because every block can be transformed into an equivalent block of the same size, satisfying these requirements. The first transformation simply introduces a new variable for each complex subexpression. The transformation for the second requirement, which is a bit tedious, is illustrated in Figure 5. Note that X_1 occurs in all right-hand-sides not guarded by a modality.

$$\begin{array}{rcl} min\{X_1 = X_1 \vee \Phi\} & \rightsquigarrow & min\{X_1 = \Phi\} \\ max\{X_1 = X_1 \vee \Phi\} & \rightsquigarrow & \mathtt{tt} \\ min\{X_1 = X_1 \wedge \Phi\} & \rightsquigarrow & \mathtt{ff} \\ max\{X_1 = X_1 \wedge \Phi\} & \rightsquigarrow & max\{X_1 = \Phi\} \end{array}$$

Figure 5: Transformation for the Second Requirement.

4.3.2 The Algorithm

Let B be a closed min-block of equations where the left-hand-side variable of the first equation X_1 represents the property to be investigated. Then we denote with X the set of all variables in B. Moreover, let $\mathcal{D} \stackrel{\text{def}}{=} 2^X \to 2^X$ be the set of all functions from 2^X to 2^X and $PT_\emptyset \in \mathcal{D}$ be the function which maps every $M \subseteq X$ to the empty set as well as PT_{Id} be the identity on $\mathcal{D}$.

Given a SPTS $\tilde{\mathcal{T}} = \{P_i\}_{i \in I}$ we now associate with each $\sigma \in \Sigma$ a function $PT_\sigma \in \mathcal{D}$ and with each transition $\stackrel{\alpha}{\to}$ a function $PT_{[\![\alpha]\!]} \in \mathcal{D}$. The function $PT_{[\![\alpha]\!]}$ represents the property transformer of the transition system which reflects the execution of $\stackrel{\alpha}{\to}$. It is defined as follows.

<u>case 1:</u> $\alpha \equiv a \in Act$: Let $M \subseteq X$ and $X_i = \Phi_i$ be some equation of B.

$$\text{Then } X_i \in PT_{[\![a]\!]}(M) \text{ iff } \begin{cases} \Phi_i = \langle a \rangle X_j & \text{and } X_j \in M \\ \Phi_i = [a] X_j & \text{and } X_j \in M \\ \Phi_i = [b] X_j & \text{and } b \neq a \end{cases}$$

<u>case 2:</u> $\alpha \equiv P_i$: $\quad PT_{[\![P_i]\!]} \stackrel{\text{def}}{=} PT_{\mathtt{start}_{P_i}}$

Now let PT_i for $i \in \{1, \ldots, n\}$ be a property transformer in $\mathcal{D}$ and $\sigma \in \Sigma$. Then the function $\Diamond^\sigma_{i=1,\ldots,n} PT_i$ is defined as follows. Given $M \subseteq X$ and the equation $X_j = \Phi_j$ of B we define $M' \stackrel{\text{def}}{=} (\Diamond^\sigma_{i=1,\ldots,n} PT_i)(M)$ by

$$X_j \in M' \text{ iff } \begin{cases} \Phi_j = A & \text{and } \sigma \in \mathcal{V}(A) \\ \Phi_j = X_k \wedge X_l & \text{and } X_k \in M' \text{ and } X_l \in M' \\ \Phi_j = X_k \vee X_l & \text{and } (X_k \in M' \text{ or } X_l \in M') \\ \Phi_j = \langle a \rangle X_k & \text{and there exists } i \in \{1, \ldots, n\} \text{ with } X_j \in PT_i(M) \\ \Phi_j = [a] X_k & \text{and } X_j \in PT_i(M) \text{ holds for all } i \in \{1, \ldots, n\} \end{cases}$$

1. iterative computation of a *property transformer* for each state class of the finite process representation. This computation works block-wise. In the case of a *min*-block it successively updates the initial "minimal" predicate transformers until the smallest fixpoint is reached. The handling of *max*-blocks is completely dual, and the global algorithm, which deals with a hierarchy of blocks is done hierarchically as described in [CS1].

   The iteration mechanism as such is essentially the same as for usual iterative algorithms, except for:

   - the handling of the procedural identifiers (non-terminals). They are dealt with by applying the currently valid approximation for their defining PTS.
   - the domain for the iteration, which is higher order here, i.e. it consists of property transformers.

   This computation is effective because we can restrict our attention to the subformulas of the property currently being investigated.

2. decision of the model-checking problem: having the predicate transformers at hand, 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 the property transformer associated with the root to the set of formulas that are valid for a deadlocking or terminating state.

4.3 The Model-Checking Algorithm

In this section we present the algorithm for *min*-blocks. This is sufficient, because the algorithm for the full alternation-free modal mu-calculus can be obtained straightforwardly along the lines of [CS1]:

- The handling of *max*-blocks is completely dual.
- The analysis of a system of nested *min*- and *max*-blocks must be done in a hierarchical fashion according to the innermost evaluation strategy.

4.3.1 Preparations

To simplify the algorithm, we assume the following structure for the (*min*-) block under consideration:

1. the right-hand-sides of blocks consist of *simple* basic formulas, which are characterized by the following grammar.

$$\Phi^{simple} ::= A \mid X \vee X \mid X \wedge X \mid \langle a \rangle X \mid [a]X$$

2. the graph, which has an arc $X_i \to X_j$ if X_j appears outside the scope of a modality on the right-hand-side of the equation for X_i, is acyclic.

valid formulas, together with its extension to systems of procedural transition systems, see Section 4.1

The effectiveness of our method is based on the observation that it is sufficient to deal with the part of a property transformer which concerns the subformulas of the property to be investigated,[5] see Section 4.2 & 4.3.

4.1 Higher Order Semantics

Let $\mathcal{T} = \langle \mathcal{S}, \mathtt{start}_{\mathcal{T}}, \mathtt{end}_{\mathcal{T}}, Act, \rightarrow \rangle$ be a simple PTS. Then we interpret $\mathcal{T}$ as the *property transformer* that yields the set of formulas which are valid at the start state, relative to the assumption that the set of propositions serving as the argument are valid at the end state. For the formal definition of the higher order semantics of $\mathcal{T}$ let RLTS denote the set of rooted LTSs. Then the semantic $[\![\mathcal{T}]\!] : 2^{\mathcal{F}} \longrightarrow 2^{\mathcal{F}}$ is defined as follows.

$$[\![\mathcal{T}]\!](M) = \{ \langle \Phi, \mathcal{B} \rangle \in \mathcal{F} \mid \forall \mathcal{T}' \in \mathrm{RLTS}.\ \mathtt{end}_{\mathcal{T}} \models_{\mathcal{T};\mathcal{T}'} M \implies \mathtt{start}_{\mathcal{T}} \models_{\mathcal{T};\mathcal{T}'} \Phi \textit{ where } \mathcal{B} \}$$

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

Theorem 4.1 (Consistency of Higher Order Semantics)
Let $\mathcal{T}$ be a simple PTS, $\mathcal{B}$ be a closed set of blocks and $\Phi \in \mathcal{F}(\mathcal{B})$. Then we have:

$$\mathtt{start}_{\mathcal{T}} \models \Phi \textit{ where } \mathcal{B} \qquad \Longleftrightarrow \qquad \langle \Phi, \mathcal{B} \rangle \in [\![\mathcal{T}]\!](\mathbf{deadlock}),$$

where **deadlock** *is the set of all propositions that are "true" at a "deadlocked state", i.e.:*

$$\mathbf{deadlock} \stackrel{\mathrm{def}}{=} \{ \langle \Phi, \mathcal{B} \rangle \mid s \models \Phi \textit{ where } \mathcal{B} \textit{ in } \mathcal{T} = \langle \{ s \}, Act, \emptyset \rangle \}.$$

The extension of the higher order semantics for simple PTSs to systems of procedural transition systems is straightforward. In this case we associate with each state class σ of P_i the property transformer induced by the fully expanded PTS of $P_i^{(\sigma)}$. This leads to the following definition.

Definition 4.2 *Let $\tilde{\mathcal{T}} = \{P_i\}_{i \in I}$ be a system of procedural transition systems and $\sigma \in \Sigma_{P_i}$ be a state class of P_i. Then we define: $[\![\sigma]\!]^{\tilde{\mathcal{T}}} \stackrel{\mathrm{def}}{=} [\![Expand_{\tilde{\mathcal{T}}}(P_i^{(\sigma)})]\!]$.*

4.2 The Methodology of the Algorithm

Our model-checker iteratively computes $[\![\sigma]\!]^{\tilde{\mathcal{T}}}$ for every state class σ of the given system of procedural transition systems and subsequently applies these property transformers to solve the model-checking problem. Thus it has the following two step structure:

[5] In terms of equational blocks $\mathcal{B}$ this means we only have to consider functions from 2^X to 2^X where X is the set of variables in $\mathcal{B}$.

Intuitively, the proposition X_2 *where* $\{B_1, B_2\}$ represents the CTL formula $AG\,AF\ P$ — "it is always the case that, eventually, P will hold" — for labeled transition systems in which $Act = \{a\}$.

3.4 The Alternation-Free Modal Mu-Calculus

In this section we establish a correspondence between the logic introduced in Sections 3.1 and 3.2 and the alternation-free modal mu-calculus, which instead of equational blocks uses the greatest and least fixpoint operators ν and μ, in order to specify unbounded behaviour. Emerson and Lei [EL] define the notion of alternation depth of a formula in the modal mu-calculus. Intuitively, the alternation depth of a formula refers to the "level" of mutually recursive greatest and least fixed-point operators. When no such mutual recursion exists, the alternation depth is one, and the formula is said to be *alternation-free*. The alternation-free fragment of the full logic is denoted by $L\mu_1$ (cf. [EL]). We have the following.

Theorem 3.2 (Expressivity)
Let $\mathcal{T}$ be a transition system, and e be an environment mapping formula variables to sets of states in $\mathcal{T}$. Then:

1. *Every formula Γ in $L\mu_1$ can be translated in time proportional to the size of Γ into a block set $\mathcal{B}$ with $[\![\Gamma]\!]e = [\![X]\!]([\![\mathcal{B}]\!]e)$ for some left-hand-side variable X of $\mathcal{B}$.*
2. *For every block set $\mathcal{B}$ and variable X there is a formula Γ in $L\mu_1$ with $[\![X]\!]([\![\mathcal{B}]\!]e) = [\![\Gamma]\!]e$.*

The required translations are straightforward. Thus we only sketch the translation from $L\mu_1$ to our logic here, in order to show that our logic is as expressive as the alternation-free modal mu-calculus. Given a $L\mu_1$ formula Φ, one constructs an equation system by generating a new equation with distinct left hand side variable for each subformula of Φ. The *parity* of these new left-hand side variables, i.e. the fact whether they are members of a min or a max block, is determined by the most recently encountered fixpoint formula.

4 Verifying Behavioural Properties

The heart of our algorithm consists of determining *property transformers* for each state class $\sigma \in \Sigma_{P_i}$ of the finite process representation under consideration. Property transformers are functions which describe the behaviour of the fully expanded transition system of P_i rooted by σ in the following way: given a set of formulas M as the argument the property transformer yields the set of formulas which are valid at σ *under the assumption* that all formulas of M are valid at the end state of P_i.

Property transformers provide a sound basis for model-checking. This is a consequence of the higher-order formulation of the semantics of simple PTSs in terms of its with respect to this lattice and therefore, according to the Tarski fixed-point theorem [Ta], has both a *greatest* fixed point, νf_E^e, and a *least* fixed point, μf_E^e. In general, these may be characterized as follows.

$$\begin{aligned} \nu f_E^e &= \bigcup\{\, \overline{S} \mid \overline{S} \subseteq f_E^e(\overline{S}) \,\} \\ \mu f_E^e &= \bigcap\{\, \overline{S} \mid f_E^e(\overline{S}) \subseteq \overline{S} \,\} \end{aligned}$$

Blocks $max\{E\}$ and $min\{E\}$ are now interpreted as *environments* in the following fashion.

$$\begin{aligned} [\![max\{E\}]\!] e &= e_{\nu f_E^e} \\ [\![min\{E\}]\!] e &= e_{\mu f_E^e} \end{aligned}$$

So $max\{E\}$ represents the "greatest" fixed point of E, while $min\{E\}$ represents the least.

We now give the semantics of a (finite) set of blocks $\mathcal{B}$ satisfying our syntactic condition. Let $B_1, \ldots, B_m$ be a topological sorting of the blocks in $\mathcal{B}$ according to the relation $\rightarrow$ defined above. Note that the syntactic restriction ensures the following: the variables that can appear on the right-hand side of an equation in B_j can only appear on the left-hand side of equations in blocks B_i with $i \leq j$, if they appear on any left-hand side at all. We now define the following sequence of environments, where e is given.

$$\begin{aligned} e_1 &= [\![B_1]\!] e \\ &\vdots \\ e_m &= [\![B_m]\!] e_{m-1} \end{aligned}$$

Then $[\![\mathcal{B}]\!] e = e_m$.

It is possible to define what it means for a state in a transition system to satisfy a formula whose variables are "bound" by a set of equations. First, we say that a basic proposition Φ is *closed* with respect to a set of blocks $\mathcal{B}$ if every variable in Φ appears on the left-hand side of some equation in some block in $\mathcal{B}$. We also refer to a set of blocks $\mathcal{B}$ as closed if each right-hand side in each block in $\mathcal{B}$ is closed with respect to $\mathcal{B}$. We shall denote with $\mathcal{F}(\mathcal{B})$ the set of all formulas closed w.r.t. $\mathcal{B}$ and with $\mathcal{F}$ the set of all pairs $\langle \Phi, \mathcal{B} \rangle$, where $\mathcal{B}$ is a closed set of blocks and $\Phi \in \mathcal{F}(\mathcal{B})$. When $\mathcal{B}$ is closed, the values assigned to the left-hand side variables of $\mathcal{B}$ in $[\![\mathcal{B}]\!] e$ do not depend on e. Thus, we have the following theorem.

Theorem 3.1 *Let $\mathcal{B}$ be a closed set of blocks, and Φ be a (basic) proposition that is closed with respect to $\mathcal{B}$. Then for any environments e and e',* $[\![\Phi]\!]([\![\mathcal{B}]\!] e) = [\![\Phi]\!]([\![\mathcal{B}]\!] e')$.

When $\mathcal{B}$ is closed with respect to itself we often omit reference to e and speak of $[\![\mathcal{B}]\!]$, and we write $s \models \Phi$ *where* $\mathcal{B}$ when Φ and $\mathcal{B}$ are closed with respect to $\mathcal{B}$ and $s \in [\![\Phi]\!][\![\mathcal{B}]\!]$.

To illustrate how properties may be formulated using sets of blocks of equations, consider the following set containing two blocks.[4]

$$\begin{aligned} &B_1 \equiv min\{X_1 = P \vee (\langle a \rangle tt \ \wedge \ [a]X_1)\} \\ &B_2 \equiv max\{X_2 = X_1 \wedge \ [a]X_2\} \end{aligned}$$

[4]Here tt is an atomic proposition that holds of every state in every labeled transition system.

in which each Φ_i is a basic formula and the X_i are all distinct. Intuitively, a block defines n mutually recursive propositions, one per variable; the precise role played by the *max* and *min* indicators will become clear in a moment. Several blocks may be used to define formulas, and the right-hand sides of an equation in one block may refer to variables appearing on the left-hand sides of equations in other blocks. In what follows we assume that all the variables that appear on the left-hand sides in a set of blocks are distinct.

We also impose the following syntactic restriction. Define $B_i \rightarrow_{\mathcal{B}} B_j$ if B_i and B_j are distinct and a left-hand-side variable in B_i appears in a right-hand-side expression of B_j. Then the "block graph" induced by $\rightarrow_{\mathcal{B}}$ must be acyclic. This ensures that there are no *alternating fixed points* [EL]. As an example, consider the block sets given in Figure 4. Block set 1 satisfies the syntactic restriction; $B_1 \rightarrow_{\mathcal{B}} B_2$ is the only edge in the block graph described above, since X_2 does not appear in B_1. Set 2, however, fails our syntactic restriction, since $B_1 \rightarrow_{\mathcal{B}} B_2$ and $B_2 \rightarrow_{\mathcal{B}} B_1$.

1.

$$\begin{array}{rcl} \{\ B_1 & \equiv & min\{X_1 = A \wedge [a]X_1 \wedge \langle a\rangle tt\} \\ B_2 & \equiv & max\{X_2 = X_1 \wedge [a]X_2\}\ \} \end{array}$$

2.

$$\begin{array}{rcl} \{\ B_1 & \equiv & max\{X_1 = X_2\} \\ B_2 & \equiv & min\{X_2 = [a](A \wedge X_1) \vee X_2\}\ \} \end{array}$$

Figure 4: Examples of Block Sets.

3.3 Semantics of Equational Blocks

To define the semantics of a set $\mathcal{B}$ of blocks, we first define the semantics of an individual block. Let E be the set of equations

$$\begin{array}{rcl} X_1 & = & \Phi_1 \\ & \vdots & \\ X_n & = & \Phi_n. \end{array}$$

Then, given a fixed environment e, we may build a function $f_E^e : (2^{\mathcal{S}})^n \rightarrow (2^{\mathcal{S}})^n$ as follows. Let $\overline{S} = \langle S_1, \ldots, S_n\rangle \in (2^{\mathcal{S}})^n$, and let $e_{\overline{S}} = e[X_1 \mapsto S_1, \ldots, X_n \mapsto S_n]$ be the environment that results from e by updating the binding of X_i to S_i. Then

$$f_E^e(\overline{S}) = \langle [\![\Phi_1]\!] e_{\overline{S}}, \ldots, [\![\Phi_n]\!] e_{\overline{S}}\rangle.$$

$(2^{\mathcal{S}})^n$ 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 system E and environment e, f_E^e is monotonic

3.1 Syntax and Semantics of Basic Formulas

Let *Var* be a (countable) set of variables, $\mathcal{A}$ a set of atomic propositions, and *Act* a set of actions. In what follows, X will range over *Var*, A over $\mathcal{A}$, and a over *Act*. Then the syntax of *basic* formulas is given by the following grammar.

$$\Phi ::= A \;|\; X \;|\; \Phi \vee \Phi \;|\; \Phi \wedge \Phi \;|\; \langle a \rangle \Phi \;|\; [a]\Phi$$

The formal semantics appears in Figure 3. It is given with respect to a labeled transition system $\langle \mathcal{S}, Act, \rightarrow \rangle$, a valuation $\mathcal{V}$ mapping atomic propositions to subsets of $\mathcal{S}$, and an environment e mapping variables to subsets of $\mathcal{S}$.

Formulas are interpreted with respect to a fixed labeled transition system $\langle \mathcal{S}, Act, \rightarrow \rangle$, a valuation $\mathcal{V} : \mathcal{A} \rightarrow 2^{\mathcal{S}}$, and an environment $e : Var \rightarrow 2^{\mathcal{S}}$.

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

Figure 3: The Semantics of Basic Formulas.

Intuitively, the semantic function maps a formula to the set of states for which the formula is "true". Accordingly, a state s satisfies $A \in \mathcal{A}$ if s is in the valuation of A, while s satisfies X if s is an element of the set bound to X in e. 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 satisfies Φ.

3.2 Syntax of Equational Blocks

By themselves, basic formulas do not provide much expressive power. The modal operators only permit one to make assertions about the states reachable from a given state via one computation step, so using only basic formulas one can only describe properties about a finite part of a systems behavior. To remedy this lack of expressiveness, formulas may also be defined using sets of *blocks* of (mutually recursive) equations. A *block* of equations has one of two forms — $min\{E\}$ or $max\{E\}$ — where E is a list of equations

$$\begin{array}{rcl}
X_1 & = & \Phi_1 \\
 & \vdots & \\
X_n & = & \Phi_n
\end{array}$$

σ' and $\mathtt{end}_{P_j}$. The fully expanded transition system of P_i with respect to $\tilde{\mathcal{T}}$, which we denote by $Expand_{\tilde{\mathcal{T}}}(P_i)$, is again rooted. Figure 2 illustrates the stepwise expansion for the encoding machine presented in Figure 1.

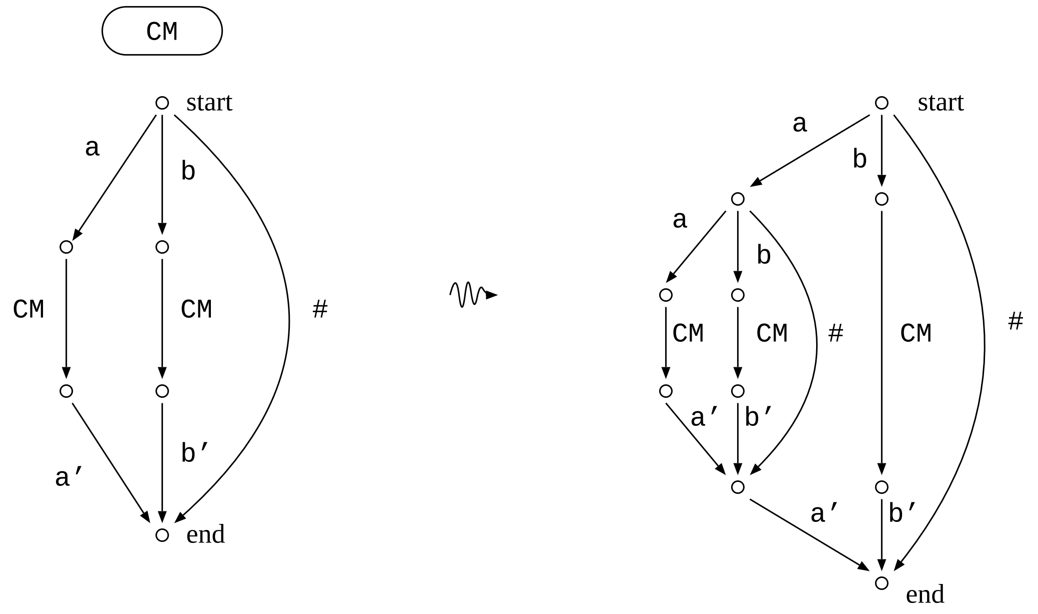


Figure 2: Expanding a Procedural Transition System

Given some state $s \in \mathcal{S}$ of a fully expanded transition system $Expand_{\tilde{\mathcal{T}}}(P_i)$, we say that s belongs to a state class $\sigma \in \Sigma_{P_j}$ if it arose as a copy of σ during the expansion. Thus a state class stands for a possibly infinite set of states of the corresponding fully expanded transition system.

Finally we introduce some notations which concern (systems of) procedural transition systems. If $\mathcal{P} = \langle \Sigma_{\mathcal{P}}, \mathtt{start}_{\mathcal{P}}, \mathtt{end}_{\mathcal{P}}, Trans, \rightarrow_{\mathcal{P}} \rangle$ is a PTS and σ is a state class of $\mathcal{P}$, we denote with $\mathcal{P}^{(\sigma)}$ the PTS $\langle \Sigma_{\mathcal{P}}, \sigma, \mathtt{end}_{\mathcal{P}}, Trans, \rightarrow_{\mathcal{P}} \rangle$ which is *rooted* by σ.

If T_1 and T_2 are two PTS, we define their sequential composition $T_1; T_2$ as the PTS

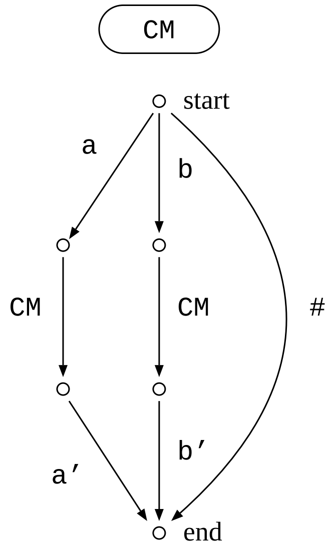
$$\langle \Sigma_{T_1} \cup \Sigma_{T_2} \setminus \{ \mathtt{start}_{T_2} \}, \mathtt{start}_{T_1}, \mathtt{end}_{T_2}, Trans_{T_1} \cup Trans_{T_2}, \rightarrow_{T_1} \cup \rightarrow_{T_2'} \rangle$$

where $\rightarrow_{T_2'}$ denotes the transition relation which results by substituting in $\rightarrow_{T_2}$ all occurrences of $\mathtt{start}_{T_2}$ by $\mathtt{end}_{T_1}$.

3 Specifying Behavioural Properties

The logic we consider may be viewed as a variant of the modal mu-calculus [Ko], or the Hennessy-Milner Logic with recursion [La]. For technical reasons, however, we represent the formula to be investigated by means of a hierarchical system of (mutually recursive) equations (cf. [CS1,CS2]). This logic is as expressive as the alternation-free modal mu-calculus, see Section 3.4.

It should be noted that $\mathcal{P}$ can well be a member of P, in which case $\mathcal{P}$ is directly recursive. If $\mathcal{P}$ contains no calls at all (P is the empty set) we say that $\mathcal{P}$ is *simple*, i.e. that $\mathcal{P}$ is a standard labeled transition system with a distinguished start and end state. In the remainder of the paper we omit reference to P if the set of calls is clear from the context.



Example: The encoding machine is a directly recursive PTS. It reads a word over the alphabet $\{ a, b \}$ until it finds the word delimiter $\#$. Then, in its second phase, it outputs the corresponding sequence of codes in reverse order.

Figure 1: The Encoding Machine

The extension to *systems of procedural transition systems* which model mutually recursive context-free processes is now straightforward.

Definition 2.3 (System of Procedural Transition Systems)
A closed system of procedural transition systems *(SPTS) $\tilde{\mathcal{T}}$ is a finite set of procedural transition systems $\{P_i\}_{i \in I}$ over $\{ P_i \mid i \in I \}$.*

The encoding machine of Figure 1 is an example of a system of procedural transition systems with only one component. We will see that this example is already sufficient to illustrate the essential features of our model-checking algorithm.

Henceforth we denote by Σ the union of all state classes of $\tilde{\mathcal{T}}$, i.e. $\Sigma \stackrel{\text{def}}{=} \bigcup_{i \in I} \Sigma_{P_i}$, and by $\rightarrow$ the union of all transition relations, i.e. $\rightarrow \stackrel{\text{def}}{=} \bigcup_{i \in I} \rightarrow_{P_i}$. Moreover, as before for labeled transition systems, we use $\sigma \stackrel{\alpha}{\rightarrow} \sigma'$ instead of $\langle \sigma, \alpha, \sigma' \rangle \in \rightarrow$, and we write $\sigma \stackrel{\alpha}{\rightarrow}$ when there is an σ' such that $\sigma \stackrel{\alpha}{\rightarrow} \sigma'$. Also the notions of *$\alpha$-derivative* and *rooted systems of procedural transition systems* $\langle \tilde{\mathcal{T}}, \mathtt{start}_{P_i} \rangle$ extend to systems of procedural transition systems straightforwardly.

A rooted system of procedural transition systems $\langle \tilde{\mathcal{T}}, \mathtt{start}_{P_i} \rangle$ serves as a finite representation of the *fully expanded transition system* of P_i with respect to $\tilde{\mathcal{T}}$, which results from successively replacing in P_i each transition $\sigma \stackrel{P_j}{\rightarrow} \sigma'$ by a copy of the corresponding procedural transition system P_j, while identifying σ and $\mathtt{start}_{P_j}$ as well as remainder of the paper 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 say that s' is an *a-derivative* of s.

Given a labeled transition system $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle$, processes are defined as *rooted transition systems*, i.e. as pairs $\langle \mathcal{T}, s \rangle$, where $s \in \mathcal{S}$ is a distinguished element, the "start state". Finally, when $\mathcal{S}$ and Act are finite, we shall say that the labeled transition system is *finite-state*.

2.1 Systems of Procedural Transition Systems

We are now going to provide a finite representation for the class of *context-free* processes which may have infinitely many states. These processes correspond to the normed processes defined by a finite set of guarded recursion equations in the signature of BPA (Basic Process Algebra) (cf. [BK]).

However, instead of using the formalism of normed processes or context-free grammars we represent context-free processes by means of *systems of procedural transition systems* which are essentially labeled transition systems whose set of actions is enlarged by names of other procedural transition systems. Intuitively, a transition labeled with such a name represents a call to the denoted procedural transition system. Thus the names of procedural transition systems correspond to the non-terminals while the actions correspond to the terminals of a context-free grammar.

Here we prefer the more dynamic procedural point of view, because it better reflects the transformational character of transition systems, which is essential for our algorithm presented in Section 4.

Definition 2.2 *A* procedural transition system (PTS) named by $\mathcal{P}$ over a set of names $P = \{ P_1, \ldots, P_n \}$ *is a quintuple* $\langle \Sigma_\mathcal{P}, \mathtt{start}_\mathcal{P}, \mathtt{end}_\mathcal{P}, Trans, \rightarrow_\mathcal{P} \rangle$, *where:*

- $\Sigma_\mathcal{P}$ *is a set of* state classes;[3]
- $\mathtt{start}_\mathcal{P} \in \Sigma_\mathcal{P}$ *is a class of* "start states"*;*
- $\mathtt{end}_\mathcal{P} \in \Sigma_\mathcal{P}$ *is a class of* "end states"*;*
- $Trans \stackrel{\text{def}}{=} Act \cup P$ *is a set of transformations, where* Act *is a set of* actions *and* P *is a set of* calls*; and*
- $\rightarrow_\mathcal{P} = \rightarrow_\mathcal{P}^{Act} \cup \rightarrow_\mathcal{P}^{P}$ *is the* transition relation*, where* $\rightarrow_\mathcal{P}^{Act} \subseteq \Sigma_\mathcal{P} \times Act \times \Sigma_\mathcal{P}$ *and* $\rightarrow_\mathcal{P}^{P} \subseteq \Sigma_\mathcal{P} \times P \times \Sigma_\mathcal{P}$*;*

Additionally $\mathcal{P}$ *must satisfy the following two restrictions:*

1. *The distinguished state class* $\mathtt{end}_\mathcal{P}$ *must be a* terminating *state class of* $\mathcal{P}$*, i.e. there does not exist any* $\alpha \in Trans, \sigma' \in \Sigma_\mathcal{P}$ *with* $\langle \mathtt{end}_\mathcal{P}, \alpha, \sigma' \rangle \in \rightarrow_\mathcal{P}$*; and*
2. $\mathcal{P}$ *must be* guarded*, i.e. that all initial transitions of* $\mathcal{P}$ *are labeled with atomic actions or formally for all* $\alpha \in Trans, \sigma' \in \Sigma_\mathcal{P}$. $\langle \mathtt{start}_\mathcal{P}, \alpha, \sigma' \rangle \in \rightarrow_\mathcal{P}$ *implies* $\alpha \in Act$.

[3]As will be explained below, members of $\Sigma_\mathcal{P}$ represent classes or sets of states in the usual sense.

The point of our algorithm is to consider a *higher order* variant of the standard iterative model-checking techniques, which instead of determining properties, determines property transformers. Altogether the model-checker has the following two step structure:

1. iterative computation of a *property transformer* for each state class of the finite process representation. These property transformers give semantics to the procedural components (or the non-terminals of the context-free representation). Intuitively they describe the set of formulas that are valid before the procedure calls (non-terminals) relative to the set of formulas that are valid afterwards. This computation is effective because we can restrict our attention to the subformulas of the property currently being investigated.

2. decision of the model-checking problem: having the predicate transformers at hand, 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 the property transformer associated with the root to the set of formulas that are valid for a deadlocking or terminating state.

The complexity of the resulting algorithm is linear in the size of the system's representation and exponential in the size of the property.[2]

The remainder of the paper develops along the following lines. Section 2 describes systems of procedural transition systems, which model context-free processes. The section following then presents our logic, while Section 4 gives our model-checking algorithm. The final section contains our conclusions and directions for future research.

2 Modelling Behaviour

In this section we introduce the notion of a procedural transition system. It will be used to finitely represent infinite state behaviours, which themselves are modelled by *labeled transition systems*:

Definition 2.1 *A* labeled transition system *(LTS) is denoted* $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle$*, 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 the* transition relation*;*

Intuitively, a labeled transition system encodes the operational behavior of a process. The set $\mathcal{S}$ represents the set of states the process may enter, and Act contains the set of actions the process may perform. The relation $\rightarrow$ describes the actions available to states and the state transitions that may result upon execution of the actions. In the

[2] In fact, it is only exponential in the weight of the property, see Section 4.4.

Model Checking for Context-Free Processes

Olaf Burkart and Bernhard Steffen
Lehrstuhl für Informatik II
RWTH Aachen
Ahornstraße 55
W-5100 Aachen
GERMANY

Abstract

We develop a model-checking algorithm that decides for a given *context-free* process whether it satisfies a property written in the alternation-free modal mu-calculus. The central idea behind this algorithm is to raise the standard iterative model-checking techniques to *higher order*: in contrast to the usual approaches, in which the set of formulas that are satisfied by a certain state are iteratively computed, our algorithm iteratively computes a *property transformer* for each state class of the finite process representation. These property transformers can then simply be applied to solve the model-checking problem. The complexity of our algorithm is linear in the size of the system's representation and exponential in the size of the property being investigated.

1 Introduction

Model-checking provides a powerful tool for the automatic verification of behavioural systems. The corresponding standard algorithms fall into two classes: the iterative algorithms (cf. [EL,CES,CS1,CS2]) and the tableaux-based algorithms (cf., e.g. [Bra,C,La,SW,Wi]). Whereas the former class usually yields higher efficiency, the latter allows local model checking (cf. [SW]). This observation has been exploited by Bradfield [Bra] for the automatic verification of infinite-state systems for a restricted class of properties, which, for example, excludes safety properties. Another approach to the verification of infinite state systems has been proposed by Baeten, Bergstra and Klop [BBK], and Hüttel and Stirling [HS], who showed that bisimulation is decidable for context-free processes.

In this paper we develop an iterative algorithm that decides the alternation-free modal mu-calculus for *context-free* processes, i.e. for processes that are given in terms of a context-free grammar, or equivalently, as mutually recursive systems of finite state-labeled transition systems.[1]

[1]Note that the (alternation-free) modal mu-calculus only allows to express structurally restricted properties. In fact, our algorithm decides essentially "regular" properties of context-free processes.