Faster Model Checking for the Modal Mu-Calculus

Rance Cleaveland* Marion Klein† Bernhard Steffen†

Abstract

In this paper, we develop an algorithm for model checking that handles the full modal mu-calculus including *alternating* fixpoints. Our algorithm has a better worst-case complexity than the best known algorithm for this logic while performing just as well on certain sublogics as other specialised algorithms. Important for the efficiency is an alternative characterisation of formulas in terms of equational systems, which enables the sharing and reuse of intermediate results.

1 Introduction

Much work in the field of automated verification has focused on finite-state transition systems (or automata) as models for system behavior [CES, CPS1, CPS2, Fe, MSGS, RRSV, RdS]. The modal mu-calculus [Ko] is a particularly useful logic for reasoning about such models; not only may a number of temporal logics for expressing system properties be translated into it [EL], but it may also be used to encode various behavioral equivalences and preorders [Ste, SI]. Thus, this logic supports algebraic as well as logic-based approaches to verification.

In this paper, we present an algorithm for determining when states in a finite-state transition system possess properties expressed in the modal mu-calculus. Our model-checking algorithm improves on the best existing methods for model checking in this logic [A, EL] while performing just as well on certain sublogics as specialized algorithms (cf. [CS1, CS2]). Important for the efficiency is an alternative characterization of formulas in terms of equational systems, which enables the sharing and reuse of intermediate results.

The remainder of the paper is organized as follows. In the next section we present the syntax and semantics of the mu-calculus, and in the section following we give an alternative, equation-based presentation of this logic. Section 4 presents our model-checking algorithm, while the subsequent section establishes its correctness and complexity. The paper closes with a detailed discussion of an example in Section 6 and some conclusions and directions for future work in Section 7.

2 Syntax and Semantics of the Mu-Calculus

This section first provides a brief overview of *labeled transition systems*, which are used as models for the mu-calculus. Then the syntax and semantics of the logic are developed.

2.1 Transition Systems

Definition 2.1 *A* labeled transition system *T is a triple $\langle \mathcal{S}, Act, \rightarrow \rangle$, 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 system. The set $\mathcal{S}$ represents the set of states the system may enter, and Act contains the set of actions the system may perform. The relation $\rightarrow$ describes the actions available to states and the state transitions that

*Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA – The author was supported by NSF Grant CCR-9014775.

†Lehrstuhl für Informatik II, RWTH-Aachen, Ahornstraße 55, W-5100 Aachen, GERMANY – Part of the work has been funded by DFG Grant Ste 537/2-1.

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 $e : Var \rightarrow 2^{\mathcal{S}}$.

$$\begin{aligned}
[\![A]\!]e &= \mathcal{V}(A) \\
[\![X]\!]e &= e(X) \\
[\![\neg\Phi]\!]e &= \mathcal{S} \setminus [\![\Phi]\!]e \\
[\![\Phi_1 \wedge \Phi_2]\!]e &= [\![\Phi_1]\!]e \cap [\![\Phi_2]\!]e \\
[\![[a]\Phi]\!]e &= \{ s \mid \forall s'.\ s \xrightarrow{a} s' \Rightarrow s' \in [\![\Phi]\!]e \} \\
[\![\nu X.\Phi]\!]e &= \bigcup \{ S' \subseteq \mathcal{S} \mid S' \subseteq [\![\Phi]\!]e[X \mapsto S'] \}
\end{aligned}$$

Figure 1: The Semantics of Formulas.

may result upon execution of the actions. In the remainder of the paper we use $s \xrightarrow{a} s'$ in lieu of $\langle s, a, s' \rangle \in \rightarrow$, and if $s \xrightarrow{a} s'$ then we say that s' is an *a-derivative* of s. Finally, we refer to a labeled transition system as *finite-state* when $\mathcal{S}$ and Act are finite.

2.2 Syntax and Semantics of Formulas

The syntax of the modal mu-calculus is parameterized with respect to a (countable) set Var of variables, a set $\mathcal{A}$ of atomic propositions, and a set Act of actions. For technical reasons we assume that $\mathcal{A}$ is closed with respect to negation: for every $A \in \mathcal{A}$ there is a $B \in \mathcal{A}$ that is semantically equivalent to the negation of A. In what follows, X will range over Var, A over $\mathcal{A}$, and a over Act. The syntax of formulas may be given by the following grammar.

$$\Phi ::= A \mid X \mid \neg\Phi \mid \Phi \wedge \Phi \mid [a]\Phi \mid \nu X.\Phi$$

The maximum fixpoint operator ν binds free occurrences of X in Φ in the usual sense. We impose an additional syntactic restriction on formulas of the form $\nu X.\Phi$: each free occurrence of X in Φ must be within the scope of an even number of negations. This requirement ensures the well-definedness of the semantics of the logic.

Let $\Phi[X := \Gamma]$ represent the formula obtained by simultaneously substituting the formula Γ for the free occurrences of the variable X in Φ. Then we may also define the usual dual operators to the ones we have presented.

$$\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2) \qquad \langle a \rangle \Phi = \neg[a](\neg\Phi) \qquad \mu X.\Phi = \neg\nu X.\neg(\Phi[X := \neg X])$$

In what follows we say that Φ' is a *proper* subformula of Φ if it is a subformula of Φ that is not Φ itself. Given a formula, its *top-level* subformulas with a certain property are defined to be those maximal proper subformulas having the property. A formula is said to be a ν-formula (μ-formula) if it has the form $\nu X.\Phi$ ($\mu X.\Phi$) for some X and Φ. We refer to a formula as *closed* if it contains no free variables and *simple* if it is fixpoint-free and contains only variables and atomic propositions as proper subformulas. For example, $X_1 \wedge A_2$ is simple, while $\langle a \rangle (X_3 \vee X_4)$ is not.

The formal semantics of formulas appears in Figure 1. It is given with respect to a finite-state 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 $\mathcal{S}$. Note that $e[X \mapsto S]$ is the environment that results by updating the binding of X to S in e.

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 $\neg\Phi$ if it does not satisfy Φ and s satisfies $\Phi_1 \wedge \Phi_2$ if it satisfies Φ_1 as well as Φ_2. The construct $[a]$ is a *modal operator*; s satisfies $[a]\Phi$ if each a-derivative of s satisfies Φ.

The syntactic restriction on the bodies of ν-formulas and the semantics of the other logical connectives ensures that semantically, the bodies give rise to monotonic functions (on the lattice sets of states) [C]. Accordingly, on the basis of the Knaster-Tarski Fixpoint Theorem [T] the semantics of $\nu X.\Phi$ is given as the greatest fixpoint of the monotonic function corresponding to Φ. In addition, for finite-state labeled transition systems the bodies of ν-formulas are continuous, and Kleene's Fixpoint Theorem then provides the following iterative characterization of the semantics. Define ϕ_i by $\phi_0 = \mathcal{S}$ and $\phi_{i+1} = [\![\Phi]\!]e[X \mapsto \phi_i]$ for $i \geq 1$. Then $[\![\nu X.\Phi]\!]e = \bigcap_{i=0}^{\infty} \phi_i$. Formula $\mu X.\Phi$ can be characterized dually as $\bigcup_{i=0}^{\infty} \hat{\phi}_i$, where $\hat{\phi}_0 = \emptyset$ and $\hat{\phi}_{i+1} = [\![\Phi]\!]e[X \mapsto \hat{\phi}_i]$. The next lemma establishes that the meaning of a closed formula does not depend on its environment.

Lemma 2.2 *Fix a finite-state transition system and valuation, and let Φ be a closed formula. Then for any environments e and e' we have: $[\![\Phi]\!]e = [\![\Phi]\!]e'$.*

The lemma holds because all variables in closed formulas are bound by a fixpoint operator, and this excludes any influence of the initial environment on the semantics of the formula. We therefore omit reference to an environment for closed formulas and write $[\![\Phi]\!]$. Finally, it is also possible to translate formulas into *positive normal form* (PNF), i.e. into a negation-free formula in which no variable is bound more than once. This is a consequence of the following lemma, where $|\Phi|$ represents the number of occurrences of operators and atomic formulas in Φ.

Lemma 2.3 *Let Φ be a closed formula in the modal μ-calculus. Then Φ can be translated into a closed formula Φ' in the logic extended with $\vee$, $\langle a \rangle$ and μ in $O(|\Phi|)$ time such that*

1) *Φ' is negation-free,* 2) $[\![\Phi]\!] = [\![\Phi']\!]$ *and* 3) $|\Phi'| \leq |\Phi|$.

The translation is done by "driving" negations inside the subformulas in the standard way following DeMorgans Laws etc, and renaming variables as appropriate. The resulting formula Φ' is not larger than Φ because of our assumptions that all free occurrences of variables in fixpoint formulas must be inside the range of an even number of negations and that the atomic propositions are closed under negation.

For notational simplicity, in what follows we only consider formulas whose top-level operator is a fixpoint operator. This is not a serious restriction, as the semantics of other formulas can be trivially determined in linear time once the semantics of the top-level fixpoint formulas have been computed.

2.3 Alternation Depth of Formulas

The complexity of the algorithm that we present in the following sections will depend on a measure on formulas called *alternation depth*. Intuitively, the alternation depth of a formula is the length of a maximal "chain" of mutually recursive greatest and least fixpoint subformulas (cf. [EL]).

Definition 2.4 (Alternation Depth of Formulas) *Let Φ be in PNF. Then the* alternation depth, *$ad(\Phi)$, of Φ is defined inductively as follows.*

- *If Φ contains closed top-level fixpoint-subformulas $\Gamma_1, \ldots, \Gamma_n$ then*

$$ad(\Phi) = max(ad(\Phi'), ad(\Gamma_1), \ldots, ad(\Gamma_n))$$

  where Φ' is obtained from Φ by substituting new atomic propositions $A_1, \ldots, A_n$ for $\Gamma_1, \ldots, \Gamma_n$.

- *If Φ contains no closed top-level fixpoint-subformulas then $ad(\Phi)$ is defined as follows.*
  - *$ad(A) = ad(X) = 0$, for any atomic proposition A and variable X.*
  - *$ad(\Phi_1 \wedge \Phi_2) = ad(\Phi_1 \vee \Phi_2) = max(ad(\Phi_1), ad(\Phi_2))$.*
  - *$ad([a]\Phi) = ad(\langle a \rangle \Phi) = ad(\Phi)$, for any action a.*

– *Let $\sigma \in \{\mu, \nu\}$, and let $\overline{\sigma}$ be the dual of σ. Then*

$$ad(\sigma X.\Phi) = max(1, ad(\Phi), 1 + ad(\overline{\sigma}X_1.\Phi_1), \ldots, 1 + ad(\overline{\sigma}X_n.\Phi_n))$$

where $\overline{\sigma}X_1.\Phi_1, \ldots, \overline{\sigma}X_n.\Phi_n$ are the top-level $\overline{\sigma}$-subformulas of Φ.

Example 2.5 For $\Phi = \nu X_1.\mu X_2.(X_1 \vee X_2 \vee \nu Y_1.\mu Y_2.\nu Y_3.(Y_1 \wedge Y_2 \wedge Y_3))$ we obtain $ad(\Phi) = 3$.

3 Equational Systems

In order to facilitate the saving and reuse of intermediate results, our model-checking algorithm works on *equational* representations of mu-calculus formulas. This section presents the syntax and semantics of the equational systems and introduces the notions of closed subsystems and alternation depth.

3.1 Syntax of Equational Systems

The systems of mutually recursive equations that we use to represent formulas are lists of the following form[1]: $\langle X_1 \circ_1 \Phi_1, \ldots, X_n \circ_n \Phi_n \rangle$ where $\circ_i \in \{\rightarrow, \leftarrow\}$. The X_i's are distinct variables, and the equation $X_i \rightarrow \Phi_i$ represents a *greatest fixpoint*, while $X_i \leftarrow \Phi_i$ represents a *least fixpoint*. Following [AC, CS1] we restrict our attention to mu-calculus formulas Φ_i that are *negation-free* and *simple*, which guarantees that every *non-atomic* right-hand-side formula has a left-hand-side variable associated with it. This facilitates the saving and reuse of intermediate results. Any equation set E may be transformed in linear time into a simple equational system E' with at most linear blow-up in size. Therefore, the model-checking algorithm presented in this paper has the same complexity for the full logic as for the simple sublogic. In what follows we refer to $X_i \rightarrow \Phi_i$ as a *max equation* with *max variable* X_i and to $X_j \leftarrow \Phi_j$ as a *min equation* with *min variable* X_j, and we associate with each left-hand-side variable a *parity* that is either *max* or *min* depending on the form of the equation. An equational system E is *closed* if all variables in a right-hand side of some equation also appear as left-hand sides in E. It should be pointed out that the order of equations is important in an equational system, owing to the presence of mutually recursive greatest and least fixpoint formulas.

Example 3.1 The following equational system E represents the formula given in Example 2.5. It can be obtained by means of the translation that will be given in Section 3.2.

$$\langle X_1 \rightarrow X_2\,,\ X_2 \leftarrow X_1 \vee X_3\,,\ X_3 \leftarrow X_2 \vee X_4\,,\ X_4 \rightarrow X_5\,,\ X_5 \leftarrow X_6\,, X_6 \rightarrow X_4 \wedge X_7\,,\ X_7 \rightarrow X_5 \wedge X_6 \rangle$$

3.2 Semantics of Equational Systems

The semantics for equational systems uses a translation from systems of equations to tuples of closed mu-calculus formulas, one for each equation. An equational system may then be interpreted as a tuple of subsets of states which arises by pointwise application of the semantic function for formulas to the component formulas.

This translation consists of the composition of two functions, B and F (for "backwards" and "forwards"), which repeatedly eliminate occurrences of free variables. Let $E = \langle X_1 \circ \Phi_1, \ldots, X_n \circ \Phi_n \rangle$ be a closed, simple equational system, and let $\overline{\Phi} = \langle \Phi_1, \ldots, \Phi_n \rangle$ consist of the right-hand sides of E. Also let $\pi_1, \ldots, \pi_n$ be the obvious projection functions. Given $\overline{\Phi}$, B produces a new tuple $\overline{\Gamma}$ of formulas by setting $\overline{\Gamma}$ to $\overline{\Phi}$ and processing each component in $\overline{\Gamma}$ as follows, beginning with $\pi_n(\overline{\Gamma})$ and working backwards.

- Replace $\pi_i(\overline{\Gamma})$ by $\mu X_i.\pi_i(\overline{\Gamma})$ (if X_i is a min-variable) or $\nu X_i.\pi_i(\overline{\Gamma})$ (if X_i is a max-variable).

[1] This form is similar to the one used by Larsen in [La].

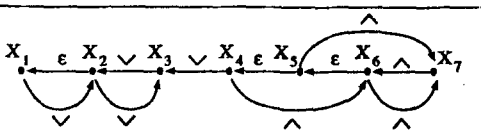

Figure 2: The Dependency Graph for Equational System E in Example 3.1.

- Substitute $\pi_i(\overline{\Gamma})$ for each free occurrence of X_i in $\pi_1(\overline{\Gamma}), \ldots, \pi_{i-1}(\overline{\Gamma})$.

Note that only $X_1, \ldots, X_{i-1}$ can appear free in $\pi_i(B(\overline{\Phi}))$; in particular, $\pi_1(B(\overline{\Phi}))$ is closed. Now F eliminates all remaining free variables: Given a tuple $\overline{\Gamma}$ of formulas, F produces a new tuple $\overline{\Delta}$ by processing each formula in $\overline{\Gamma}$ in the order of the indices as follows: substitute $\pi_i(\overline{\Delta})$ for each free occurrence of X_i in $\pi_{i+1}(\overline{\Delta}), \ldots, \pi_n(\overline{\Delta})$. The semantics of E can now be given as follows.

Definition 3.2 (Semantics of Equational Systems) *Let E be a closed, simple system of n equations, and let $\overline{\Phi}$ be the tuple of right-hand sides of E. Also let $\langle \Delta_1, \ldots, \Delta_n \rangle = F \circ B(\overline{\Phi})$. Then $[\![E]\!] = \langle [\![\Delta_1]\!], \ldots, [\![\Delta_n]\!] \rangle$.*

The connection between equational systems and the mu-calculus can be made explicit by providing translations back and forth. **trans**^{a}, translating equational systems into formulas, is straightforward in terms of F and B: **trans**$^a(E) = \pi_1(F \circ B(\overline{\Phi}))$, where $\overline{\Phi}$ consists of the right-hand sides of E. Given a mu-calculus formula Φ in PNF, the function **trans** builds an equational system by recursing through Φ, adding a new equation at the end of the last of the already generated equations for each subformula of Φ. The parity of a new added left-hand-side variable is determined by the most recently encountered fixpoint operator. As an example, consider the formula and the equational system given in Example 2.5 and 3.1, respectively. Here, the application of **trans** to Φ yields E.

Obviously, **trans** works in linear time as every subformula of Φ is investigated exactly once. Moreover, the number of equations in the resulting simple equational system E_Φ is less than or equal to the size of the formula Φ, because every subformula of Φ is transformed into at most one equation. A detailed account of these translations can be found in [CDS].

Instead of solving the model-checking problem directly for a given formula Φ we solve it on the equational system E_Φ that is gained by the translation given above. The following theorem establishes the correctness of this approach.

Theorem 3.3 *Let Φ be a closed PNF formula and $E_\Phi =$* **trans**(Φ). *Then, $[\![\Phi]\!] = \pi_1([\![E_\Phi]\!])$.*

3.3 Graph Representation of Equational Systems

In this section we introduce a graph representation of equational systems that will be used to determine the *closed subsystems* of equational systems and to define the notion of *alternation depth*. Let E be an equational system. Then its dependency graph G_E is an edge-labeled graph with one node for each left-hand-side variable in E and edges defined as follows, where $i \neq j$.

- $X_i \xrightarrow{l} X_j$ if for some Φ either $X_j \Leftrightarrow X_i \, l \, \Phi$ or $X_j \Leftrightarrow \Phi \, l \, X_i$ is an equation in E for $l \in \{\vee, \wedge\}$.
- $X_i \xrightarrow{l} X_j$ if $X_j \Leftrightarrow l \, X_i$ is in E for $l \in \{\langle a \rangle, [a]\}$.
- $X_i \xrightarrow{\varepsilon} X_j$ if $X_j \Leftrightarrow X_i$ is in E.

Intuitively, there is an edge from X_i to X_j if the meaning of X_i directly influences the meaning of X_j. In what follows, we write $X_i \rightarrow X_j$ if there is an edge in G_E from X_i to X_j and $X_i \rightarrow^* X_j$ if there is a path from X_i to X_j in G_E. As an example, the graph for the equational system in Example 3.1 appears in Figure 2.

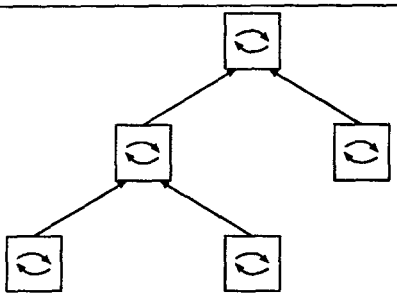

Figure 3: The Dependencies between and within the Closed Subsystems.

Theorem 3.4 *Let $E = \langle X_1 \Leftrightarrow \Phi_1, \ldots, X_n \Leftrightarrow \Phi_n \rangle$ be an equational system. Its dependency graph G_E can be constructed in $O(|E|)$ time, and it contains n vertices and no more than $2n$ edges.*

Let C be a sublist of E. Then we refer to the subgraph of G_E induced by C as G_C. Also, we write $X_i \dashrightarrow_{C,k} X_j$ if $X_i \dashrightarrow X_j$ is an edge in G_C with $i \geq k$ and $j \geq k$. These notions are used in Section 3.5.

3.4 Closed Subsystems of Equational Systems

In analogy with the notion of closed subformulas, we develop the notion of *closed subsystems* of equational systems; these turn out to be essential in order for us to achieve the desired complexity for our model-checking algorithm.

From the definition of the dependency graph G_E, if two variables X_i and X_j are such that $X_i \dashrightarrow^* X_j$ and $X_j \dashrightarrow^* X_i$, it follows that the semantics of X_i affects that of X_j, and vice versa. When this is the case we say that X_i and X_j are *mutually dependent*, since any change to the semantics of one may induce a change in the other. On the other hand, if $X_i \dashrightarrow^* X_j$ but $X_j \not\dashrightarrow^* X_i$, then changes to X_i affect X_j, but not vice versa. In this case we say that there is a *hierarchical* dependency from X_i to X_j, since once the semantics of X_i is computed future changes to X_j cannot affect it.

In graph-theoretic terms, when $X_i \dashrightarrow^* X_j$ and $X_j \dashrightarrow^* X_i$, then X_i and X_j belong to the same *strongly connected component* of G_E.[2] Within a strongly connected component each pair of variables is mutually dependent, while there can exist at most a hierarchical dependence between two variables in distinct strongly connected components. This suggests the following strategy for computing $[\![E]\!]$:

1. Build the condensation graph, G_C, of G_E. (Recall that the condensation graph of G is a graph having the strongly connected components G_i of G as its vertices, with an edge $G_i \rightarrow G_j$ defined if G_i and G_j are distinct and there are nodes $V_i \in G_i$, $V_j \in G_j$, such that $V_i \rightarrow V_j$ is an edge in G.) Note that G_C is acyclic.

2. Topologically sort G_C into $G_m, \ldots, G_1$. (Here G_m is a "source" node in G_C; we have elected to number it m so that, in general, higher-numbered variables belong to higher-numbered components.) Notice that if there is an edge from G_i to G_j then $i > j$.

3. For each G_i, generate a *closed subsystem* C_i containing the equations from E whose left-hand sides are in G_i. These equations are modified by replacing each occurrence of X_j that is not a left-hand side in G_i by a new atomic proposition A_j; this ensures that C_i is closed. Note that if X_j is in component G_k then $k < i$.

[2] Recall that a strongly connected component of a graph is a maximal subset $\mathcal{V}$ of vertices having the property that $V_i \rightarrow^* V_j$ and $V_j \rightarrow^* V_i$ for any $V_i, V_j \in \mathcal{V}$.

4. Beginning with C_m, process each C_i in turn.

As an example, consider E in Example 3.1 with its dependency graph G_E shown in Figure 2. As there are two strongly connected components of G_E we get two closed subsystems:

$$C_1 \;=\; < X_1 \to X_2\,,\; X_2 \leftarrow X_1 \vee X_3\,,\; X_3 \leftarrow X_2 \vee A_4 >$$

$$C_2 \;=\; < X_4 \to X_5\,,\; X_5 \leftarrow X_6\,,\; X_6 \to X_4 \wedge X_7\,,\; X_7 \to X_5 \wedge X_6 >$$

Note that each C_l is closed and that each left-hand-side variable X_i of E appears as a left-hand side in exactly one of the C_l. Also notice that the construction ensures that if a new atomic proposition A_i appears in a right-hand side in C_j, then X_i must appear as a left-hand side in some C_l with $l > j$. Consequently, we may define the semantics of A_i as follows. Let C_l be the closed subsystem containing X_i as a left-hand side, and let k be the index of X_i in C_l. Then $[\![A_i]\!] = \pi_k([\![C_l]\!])$. The following theorem shows that this transformation of E into $C_1 \ldots C_m$ is, in a certain sense, semantics-preserving.

Theorem 3.5 *The closed subsystems $C_1, \ldots, C_m$ of an equational system E can be determined in $O(|E|)$ time. Furthermore, if X_i is the k^{th} left-hand side in C_l, then $\pi_i([\![E]\!]) = \pi_k([\![C_l]\!])$.*

In our example we have $[\![E]\!] = \langle \pi_1[\![C_1]\!], \pi_2[\![C_1]\!], \pi_3[\![C_1]\!], \pi_1[\![C_2]\!], \pi_2[\![C_2]\!], \pi_3[\![C_2]\!], \pi_4[\![C_2]\!] \rangle$.

3.5 Alternation Depth of Equational Systems

We close this section by defining the notion of *alternation depth* of an equational system. It will turn out that this notion is consistent with the one given for formulas (cf. Theorem 3.8), and therefore we may use the same notation.

To define the alternation depth we first introduce the notion of *nesting depth* of equations that reflects the length of the chain of mutually depending *min* and *max* equations within a closed subsystem.

Definition 3.6 (Nesting Depth of Equations)
Let $E = \langle X_1 \Leftrightarrow \Phi_1, \ldots, X_n \Leftrightarrow \Phi_n \rangle$ be an equational system with its closed subsystems $C_1, \ldots, C_m$. Furthermore, assume $\sigma \in \{max, min\}$ and $\overline{\sigma}$ to be the dual parity. Then the nesting depth *of the equation with left-hand side X_i having parity σ and belonging to C_l is given by:*

$$\begin{aligned} nd(X_i, C_l) \;=\; max(1,\; & max\{\, nd(X_j, C_l) \mid X_j \dot{\to}_{C_l,i}{}^{*} X_i \text{ and } X_j \text{ has parity } \sigma \,\}, \\ & max\{\, 1 + nd(X_j, C_l) \mid X_j \dot{\to}_{C_l,i}{}^{*} X_i \text{ and } X_j \text{ has parity } \overline{\sigma} \,\}) \end{aligned}$$

The nesting depth of the closed subsystem C_l is defined as $nd(C_l) = max\{nd(X_i, C_l) | X_i \Leftrightarrow \Phi_i \in C_l\}$.

The alternation depth of an equational system is now defined as the maximal nesting depth of its closed subsystems.

Definition 3.7 (Alternation Depth of Equational Systems)
Let $E = \langle X_1 \Leftrightarrow \Phi_1, \ldots, X_n \Leftrightarrow \Phi_n \rangle$ be an equational system with closed subsystems $C_1, \ldots, C_m$. Then the alternation depth *of E is given by* $ad(E) = max\{nd(C_l) | 1 \leq l \leq m\}$.

Example: As shown already, the equational system E presented in Example 3.1 has two closed subsystems, and we have: $nd(X_3, C_1) = nd(X_2, C_1) = 1$ and $nd(X_1, C_1) = 2$, thus $nd(C_1) = 2$ and $nd(X_7, C_2) = nd(X_6, C_2) = 1$, $nd(X_5, C_2) = 2$ and $nd(X_4, C_2) = 3$, thus $nd(C_2) = 3$. Therefore $ad(E) = 3$.

We say that an equational system E is *alternation-free* if $ad(E) = 1$. The consistency of the notions of alternation depth for formulas and equational systems is a consequence of the following theorem.

Theorem 3.8 *Let Φ be a closed PNF formula with $ad(\Phi) \geq 1$ and $E_\Phi = \mathbf{trans}(\Phi)$ be the corresponding equational system. Then $ad(\Phi) = ad(E_\Phi)$.*

The left-hand-side variables of a closed subsystem of an equational system can be partitioned into *nesting levels*, which are used to guide the fixpoint computation.

Definition 3.9 (Nesting Levels) *Let $E = \langle X_1 \sigma_1 \Phi_1, \ldots, X_n \sigma_n \Phi_n \rangle$ be an equational system with closed subsystems $C_1, \ldots, C_m$. Then the set of variables belonging to a closed subsystem C_l is partitioned into* nesting levels *by $E_{l,i} = \{ X_j \mid nd(X_j, C_l) = i \}$ for $1 \leq i \leq nd(C_l)$.*

Given a nesting level $E_{l,i}$ we call the nesting level $E_{l,j}$ *lower* if $j < i$ and *higher* if $j > i$. Each nesting level consists of at most two *blocks* of equations, where a block consists entirely of min or of max equations.

Theorem 3.10 *Given an equational system E:*

1. *Alternation-freedom can be established in $O(|E|)$ time.*
2. *The nesting levels can be determined in $O(|E|^2)$ time.*

4 The Model-Checking Algorithm

In this section we present a model-checking algorithm that, given an equational system E and a transition system $T = \langle S, Act, \rightarrow \rangle$, computes $[\![E]\!]$. Due to space limitations, we only sketch an outline of the algorithm; the interested reader is referred to [CDS] for a fuller discussion of the details.

As with the algorithms in [AC, CS1, CS2], our algorithm is bit-vector-based. Each state in S has a bit vector whose i^{th} entry indicates whether or not the state belongs to the set associated with X_i in the current stage of the analysis. These bit-vectors represent the current approximation $\langle S_1, \ldots, S_n \rangle \in (2^S)^n$ to $[\![E]\!]$ during model checking as follows: $s \in S_i$ if and only if $s.X[i]$ is true, for $1 \leq i \leq n$.

Given E, the algorithm works by first determining the closed subsystems $C_1 \ldots C_m$ of E. It then processes each C_l in turn, beginning with C_m and ending with C_1; $[\![C_l]\!]$ is computed and stored in the relevant bit-vector components, and then the atomic predicates whose semantics depend on left-hand sides in C_l have their semantics initialized. The algorithm terminates after C_1 is completed. Given that each $[\![C_l]\!]$ is computed properly, correctness follows from Theorem 3.5.

At the heart of the algorithm is the computation of $[\![C_l]\!]$ for a closed subsystem C_l. This processing proceeds in two phases. During the first phase, bit-vectors are initialized such that components corresponding to *max variables* are set to *true* and components corresponding to *min variables* are set to *false*. In the second phase, the nesting levels of C_l are repeatedly analyzed, beginning with the lowest level, $E_{l,1}$, and proceeding up to $E_{l,nd(C_l)}$. To process a nesting level, the algorithm essentially invokes a variant of the alternation-free model-checking algorithm given in [CS2]. Bit-vector annotations are changed until appropriate fixed points are reached; in addition, if changing a bit-vector component in one variable also causes a change in the semantics of a variable in a lower nesting level, then the lower nesting levels that are affected must be re-initialized and recomputed. The processing of a nesting level is finished when consistency is reached with all lower levels. Then, the next higher level is begun.

In this computation of $[\![C_l]\!]$, one may identify two flows of information.

- **The flow of assumptions:** Our algorithm may be seen as "assumption based": during the computation of a fixpoint for equations in a nesting level, the variables in higher nesting levels are treated essentially as propositional constants in that their meaning is fixed. Thus, the *assumption flow* proceeds from $E_{l,nd(C_l)}$ down to $E_{l,1}$.

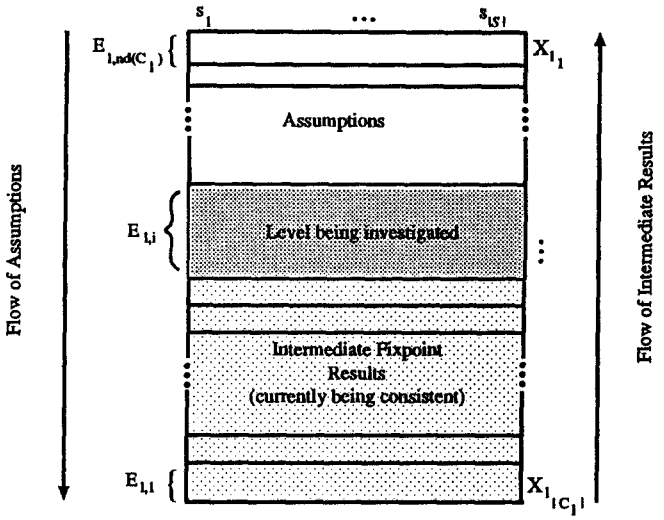

Figure 4: The Flows of Information in a Closed Subsystem C_l.

- **The propagation of intermediate results:** Fixpoints are computed from lower to higher nesting levels. Therefore, the *computation flow* proceeds in the direction opposite to that of the flow of assumptions, as intermediate results computed in one level may affect the results for higher levels.

In this view, the need for recomputing values in lower nesting levels when a higher nesting level changes becomes apparent: the computation of the lower level was based on a wrong assumption.

The two flows of information are illustrated in Figure 4, where the box represents the current approximation of the semantics of C_l with the bit-vectors corresponding to columns through the levels.

Three observations are exploited in order to achieve the complexity stated in the next section.

1. The partitioning of the equational system E into closed subsystems $C_1, \ldots, C_m$ ensures that once $[\![C_l]\!]$ is computed, it cannot be affected by the analysis of subsequent closed subsystems.
2. Within a given closed subsystem C_l the nesting levels are treated exactly as in the (linear-time [CS2]) *alternation-free* case each time their fixpoint is computed.
3. Computing consistency of the lowest, and most often recomputed, nesting level $E_{l,1}$ is less expensive than for the higher levels as $E_{l,1}$ does not give rise to resetting and recomputation of lower levels and also need not account for the new values that resetting and recomputing lower levels can give rise to.

The full structure of the model-checking algorithm is given in [CDS]; Section 6 contains an example illustrating our technique.

5 Correctness and Complexity

The correctness of the algorithm rests on the observation that our algorithm computes $[\![C_l]\!]$ component-wise according to the semantic definition of formulas by representing the environment in the bit vectors. Together with Theorem 3.5 this enables us to prove the following theorem (cf. [CDS]).

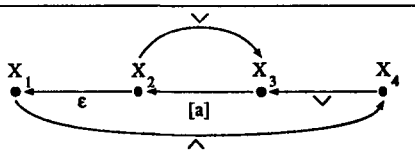

Figure 5: The Dependency Graph.

Theorem 5.1 (Correctness) *Let $\mathcal{T} = \langle S, Act, \rightarrow \rangle$ be a labeled finite-state transition system and $E = \langle X_1 \Leftarrow \Phi_1, \ldots, X_n \Leftarrow \Phi_n \rangle$ be a closed, simple equational system. Then the model-checking algorithm terminates with a bit-vector annotation that represents $[\![E]\!]$.*

The following theorem states our complexity result, where $|\mathcal{T}| = |S| + |\rightarrow|$ and $|E|$ is the number of equations in E. A complete proof is given in [CDS].

Theorem 5.2 (Complexity) *Let $E = \langle X_1 \Leftarrow \Phi_1, \ldots, X_n \Leftarrow \Phi_n \rangle$ be a simple, closed equational system with $ad(E) \geq 1$, and $\mathcal{T} = \langle S, Act, \rightarrow \rangle$ be a finite-state transition system. Then the worst-case time complexity of the model-checking algorithm is*

$$O\left(|\mathcal{T}| * |E| * \left(|S| * \frac{|E|}{ad(E)}\right)^{ad(E)-1}\right)$$

6 An Example

In this section we illustrate our algorithm with an example taken from [SW]. Consider the formula $\Phi = \nu Z.\mu Y.[a]((A \wedge Z) \vee Y)$ having alternation depth 2. The semantics of Φ with respect to a transition system $\mathcal{T}$ is the set of states for which A holds infinitely often on all a-paths. Its corresponding equational system

$$E \;=\; < X_1 \rightarrow X_2 \,,\; X_2 \leftarrow [a]X_3 \,,\; X_3 \leftarrow X_4 \vee X_2 \,,\; X_4 \leftarrow A \wedge X_1 >$$

only has the trivial closed subsystem consisting of two nesting levels: E_1 holding the last three equations, and E_2 holding the first equation. The dependency graph is shown in Figure 5.

The transition system $\mathcal{T}$ we want to investigate is the triple $\langle S, Act, \rightarrow \rangle$, where $S = \{s, t, u, v\}$, $Act = \{a\}$ and the transition relation has six elements: $s \xrightarrow{a} s$, $s \xrightarrow{a} t$, $t \xrightarrow{a} u$, $u \xrightarrow{a} s$, $u \xrightarrow{a} v$ and $v \xrightarrow{a} v$.

The valuation is given by $\mathcal{V}(A) = \{t, u, v\}$; so states t, u and v satisfy A, but s does not. Besides the bit-vectors $s.X[1..4]$, $t.X[1..4]$, $u.X[1..4]$ and $v.X[1..4]$ we need some auxiliary data structures for investigating the levels (cf. [CS2]): the counters $s.C[1..4]$, $t.C[1..4]$, $u.C[1..4]$ and $v.C[1..4]$, where $x.C[i]$ maintains a count of the number of components $y.X[j]$ that may change until $x.X[i]$ must change; and the array of worklists $M[1..4]$, where $M[i]$ holds the states the changes to whose i^{th} bit-vector components have yet to be propagated. The states also contain fields recording whether they satisfy the atomic formula A; so $s.A = ff$, while $t.A = u.A = v.A = tt$. Note that X_1 is a *max* variable initialized with *true* for all states and X_2, X_3 and X_4 are *min* variables accordingly initialized with *false*. In what follows we highlight the changes made to the data structure step by step. Note in particular the change of intermediate results in E_1 because of changing assumptions in E_2.

- Computing a fixpoint over the lowest level E_1 (containing X_2 to X_4) starts with the following initialization of the bit vectors, counters and worklists.

	s	t	u	v
X_1	tt	tt	tt	tt
X_2	ff	ff	ff	ff
X_3	ff	ff	ff	ff
X_4	ff	tt	tt	tt

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	1	0	0	0

$M[\emptyset, \emptyset, \emptyset, \{t, u, v\}]$

The influence of the states in the worklist is determined. First, t, u, v are successively deleted from $M[4]$ and $X_4 \overset{\vee}{\rightarrow} X_3$ is processed. Second, t, u, v are successively deleted from $M[3]$ and $X_3 \overset{[a]}{\rightarrow} X_2$ is processed. This provides the following intermediate results:

	s	t	u	v
X_1	tt	tt	tt	tt
X_2	ff	ff	ff	ff
X_3	ff	$\boxed{tt}$	$\boxed{tt}$	$\boxed{tt}$
X_4	ff	tt	tt	tt

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	1	0	0	0

$M[\emptyset, \emptyset, \{t, u, v\}, \emptyset]$

and

	s	t	u	v
X_1	tt	tt	tt	tt
X_2	ff	$\boxed{tt}$	ff	$\boxed{tt}$
X_3	ff	tt	tt	tt
X_4	ff	tt	tt	tt

	s	t	u	v
C_1	/	/	/	/
C_2	$\boxed{1}$	$\boxed{0}$	$\boxed{1}$	$\boxed{0}$
C_3	/	/	/	/
C_4	1	0	0	0

$M[\emptyset, \{t, v\}, \emptyset, \emptyset]$

Now t and v are successively deleted from $M[2]$ and $X_2 \overset{\vee}{\rightarrow} X_3$ is processed. As $t.X[3]$ and $v.X[3]$ are already *true* the bit-vectors remain unchanged and the worklists for E_1 are empty.

- On the next nesting level E_2 the fixpoint computation detects an inconsistency for s and an inconsistency for u as $s.X[1] = u.X[1] = tt$ but $s.X[2] = u.X[2] = ff$ and $X_2 \overset{c}{\rightarrow} X_1$. Thus $s.X[1]$ and $u.X[2]$ are set to *false* and E_1 has to be reset and recomputed accordingly.
- The recomputation of E_1 taking the new assumptions into account starts with the initialization shown on the left and computes the fixpoint shown on the right.

	s	t	u	v
X_1	$\boxed{ff}$	tt	$\boxed{ff}$	tt
X_2	ff	ff	ff	ff
X_3	ff	ff	ff	ff
X_4	ff	tt	$\boxed{ff}$	tt

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	$\boxed{2}$	0	$\boxed{1}$	0

$M[\emptyset, \emptyset, \emptyset, \{t, v\}]$

and

	s	t	u	v
X_1	ff	tt	ff	tt
X_2	ff	ff	ff	$\boxed{tt}$
X_3	ff	$\boxed{tt}$	ff	$\boxed{tt}$
X_4	ff	tt	ff	tt

	s	t	u	v
C_1	/	/	/	/
C_2	$\boxed{1}$	1	$\boxed{1}$	$\boxed{0}$
C_3	/	/	/	/
C_4	2	0	1	0

$M[\emptyset, \emptyset, \emptyset, \emptyset]$

- Again computing the fixpoint over E_2 an inconsistency is detected as $t.X[1] = tt$ but $t.X[2] = ff$ and $X_2 \overset{c}{\rightarrow} X_1$. Thus $t.X[1]$ is set to *false* and E_1 is reset and recomputed, providing the following results for initialization (left) and fixpoint computation (right):

	s	t	u	v
X_1	ff	$\boxed{ff}$	ff	tt
X_2	ff	ff	ff	ff
X_3	ff	ff	ff	ff
X_4	ff	$\boxed{ff}$	ff	tt

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	2	$\boxed{1}$	1	0

$M[\emptyset, \emptyset, \emptyset, \{v\}]$

and

	s	t	u	v
X_1	ff	ff	ff	tt
X_2	ff	ff	ff	$\boxed{tt}$
X_3	ff	ff	ff	$\boxed{tt}$
X_4	ff	ff	ff	tt

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	$\boxed{1}$	$\boxed{0}$
C_3	/	/	/	/
C_4	2	1	1	0

$M[\emptyset, \emptyset, \emptyset, \emptyset]$

Finally, E_2 is shown to be consistent, the algorithm terminates, and we obtain $[\![\Phi]\!] = \{v\}$, as $v.X[1] = tt$ and the first bit-vector component of all other states is *false*. This reflects one's intuition about the formula, because v is the only state for which A is infinitely often satisfied along all a-paths.

7 Conclusions and Future Work

In this paper, we have presented an algorithm for model checking that handles the full modal mu-calculus including *alternating* fixed points. The algorithm extends the one given in [CS2] for an alternation-free logic. Central is the new complexity result:

$$O\left(|T| * |E| * \left(|S| * \frac{|E|}{ad(E)}\right)^{ad(E)-1}\right)$$

which improves even on our conjecture ([CS2]):

- Instead of being exponential in the full size of the transition system it is only exponential in the number of its states. This saves a quadratic blow-up in the worst case.
- Instead of being exponential in the full size of the formula, it is only exponential in $\frac{|E|}{ad(E)}$, which is important for formulas with high alternation depth.

In [A] Andersen sketches an $O(|S| * |T|^{ad(\Phi)-1} * |\Phi|^{ad(\Phi)})$ algorithm for the full mu-calculus, which improves on Emerson and Lei's result, $O((|T| * |E|)^{ad(E)+1})$. Andersen's algorithm differs from ours in that it is tailored to the mu-calculus structure rather than systems of equations, where properties can be expressed much more concisely. In the worst case, his formalizations are exponentially larger than ours, because equational systems allow to compactly represent common subexpressions. This generality, however, requires a much more involved algorithm. Nevertheless, we were able to prove a stronger complexity result, even with respect to the more compact representations. Our algorithm will be implemented as an extension of the Concurrency Workbench [CPS1, CPS2].

References

[A] Andersen, H. "Model Checking and Boolean Graphs." *Proc. of ESOP '92*, LNCS 582, 1992.

[AC] Arnold, A., and P. Crubille. "A Linear Algorithm To Solve Fixed-Point Equations on Transition Systems." *Information Processing Letters*, v. 29, 30 September 1988, pp. 57–66.

[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.

[CDS] Cleaveland, R., M. Klein and B. Steffen. "Faster Model Checking for the Modal Mu-Calculus." In *Technical Report RWTH Aachen Nr. 91-29*, Fachgruppe der Informatik, 1991.

[CPS1] Cleaveland, R., J. Parrow and B. Steffen. "The Concurrency Workbench." In *Proceedings* CAV'89, LNCS 407, 1989.

[CPS2] Cleaveland, R., J. Parrow and B. Steffen. "A Semantics-based Verification Tool for Finite-State Systems", In *Proceedings of the Ninth International Symposium on Protocol Specification, Testing, and Verification.* North-Holland, 1989.

[CS1] Cleaveland, R. and B. Steffen. "Computing Behavioural Relations, Logically." In Proceedings ICALP '91, LNCS 510, 1991.

[CS2] Cleaveland, R. and B. Steffen. "A Linear-Time Model Checking Algorithm for the Alternation-Free Modal Mu-Calculus." In Proceedings CAV '91, LNCS 575, 1991.

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

[Fe] Fernandez, J.-C. *Aldébaran: Une Système de Vérification par Réduction de Processus Communicants.* Ph.D. Thesis, Université de Grenoble, 1988.

[K] Kleene, S. C. "Introduction to Metamathematics", North Holland, 1952.

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

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

[MSGS] Malhotra, J., Smolka, S.A., Giacalone, A. and Shapiro, R. "Winston: A Tool for Hierarchical Design and Simulation of Concurrent Systems." In *Proceedings of the Workshop on Specification and Verification of Concurrent Systems*, Univ. of Stirling, Scotland, 1988.

[RRSV] Richier, J., Rodriguez, C., Sifakis, J. and Voiron, J.. "Verification in XESAR of the Sliding Window Protocol." In *Proceedings of the Seventh IFIP Symposium on Protocol Specification, Testing, and Verification*, 1987, North-Holland.

[RdS] Roy, V. and R. de Simone. "Auto/Autograph." In *Proceedings*, CAV'90, LNCS 531, 1990.

[Ste] Steffen, B.U. "Characteristic Formulae." In *Proceedings ICALP*, LNCS 372, 1989.

[SI] Steffen, B.U., and A. Ingólfsdóttir. "Characteristic Formulae for CCS with Divergence." To appear in *Information and Computation.*

[SW] Stirling, C. and D. Walker. "Local Model Checking in the Modal Mu-Calculus." In *Proceedings of TAPSOFT '89*, LNCS 351, 1989.

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