Faster Model Checking for the Modal μ-Calculus

Rance Cleaveland*
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
USA

Marion Klein†
Bernhard Steffen
Lehrstuhl für Informatik II
RWTH-Aachen
Ahornstraße 55
W-5100 Aachen
GERMANY

August 9, 1993

Abstract

In this paper, we develop a model-checking algorithm for the full modal μ-calculus with *alternating* fixpoints. Our algorithm has a better worst-case complexity than the best known algorithm for this logic while performing as well on various sublogics as more specialized algorithms. The approach relies on an alternative characterization of formulas in terms of equational systems that enable the sharing and reuse of intermediate results.

1 Introduction

Much work concerning 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 μ-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.

---

*The author was supported by NSF/DARPA Grant CCR-9014775, NSF Grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, and NSF Young Investigator Award CCR-9257963.

†Part of the work has been funded by DFG Grant Ste 537/2-1.

In this paper, we present an algorithm for determining when states in a finite-state transition system possess properties expressed in the modal μ-calculus. Our model-checking algorithm improves on the best existing methods for model checking in this logic [A, EL] while performing as well on various sublogics as more specialized algorithms (cf. [CS1, CS2]). Important for the efficiency is an alternative characterization of formulas in terms of *equational systems* that enables the sharing and reuse of intermediate results during fixpoint computation which is guided by *reduced alternation depth*, a notion that improves on the commonly used notion alternation depth introduced by [EL].

The remainder of the paper is organized as follows. In the next section we introduce basic results from fixpoint theory that will be used in the rest of the paper. Section 3 presents the syntax and semantics of the μ-calculus, and in the section following we give an alternative, equation-based presentation of this logic. Section 5 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 7 and some conclusions and directions for future work in Section 8.

2 Lattices and Fixpoints

The μ-calculus includes a general mechanism for defining propositions recursively. The semantics of these propositions is typically given using results taken from lattice theory on the existence of fixpoints to certain kinds of formulas. Accordingly, in this section we review these results and introduce other notions that will be used in the rest of the paper.

2.1 Background

Let L be a set, and let $\sqsubseteq \subseteq L \times L$ be partial order on L, where a partial order is a reflexive, antisymmetric and transitive relation. Then $\langle L, \sqsubseteq \rangle$ is a *lattice* if every pair of elements $l_1, l_2 \in L$ has a least upper bound $l_1 \sqcup l_2 \in L$ and greatest lower bound $l_1 \sqcap l_2 \in L$ with respect to $\sqsubseteq$. $\langle L, \sqsubseteq \rangle$ is a *complete lattice* if every subset $L' \subseteq L$ has a least upper bound $\sqcup L' \in L$ and a greatest lower bound $\sqcap L' \in L$. Note that every complete lattice has a maximum element $\top = \sqcup \emptyset$ and minimum element $\bot = \sqcap \emptyset$, and that every finite lattice is complete.

A function $f \in L^L$ is *monotonic* if for every $l_1, l_2 \in L$ such that $l_1 \sqsubseteq l_2$, we have that $f(l_1) \sqsubseteq f(l_2)$. In this paper we use the following result, which is a corollary of a more general theorem proved by Tarski and Knaster [T].

Theorem 2.1 *Let $\langle L, \sqsubseteq \rangle$ be a complete lattice, with $f \in L^L$ monotonic. Then f has a greatest fixpoint νf and least fixpoint μf defined as follows.*

$$\begin{aligned} \nu f &= \sqcup \{\, l \in L \mid l \sqsubseteq f(l) \,\} \\ \mu f &= \sqcap \{\, l \in L \mid f(l) \sqsubseteq l \,\} \end{aligned}$$

As an example of a complete lattice, consider the set 2^S of subsets of S ordered by set inclusion. Then $\langle 2^S, \subseteq \rangle$ is a complete lattice, with $\sqcup \mathcal{S} = \bigcup \{\, S' \in \mathcal{S} \mid S' \in \mathcal{S} \,\}$ and $\sqcap \mathcal{S} = \bigcap \{\, S' \in \mathcal{S} \mid S' \in \mathcal{S} \,\}$. Note that in this complete lattice, $\bot = \emptyset$ and $\top = S$.

2.2 A Complete Lattice of Mappings

In this paper we will have occasion to use mappings whose domains are sets and codomains are lattices. In this section we show that certain collections of such mappings form lattices, and we give alternative characterizations of the fixpoints of monotonic functions over these lattices that will prove useful later.

We begin by introducing some notation. Let L and X be sets. Then the set L^X consists of all functions mapping elements of X to elements of L. We sometimes represent L^X as $X \rightarrow L$. If $f \in L^X$ and $f' \in L'^{X'}$, then we write $f[f']$ to represent the function in $(L \cup L')^{(X \cup X')}$ defined as follows.

$$(f[f'])(x) \quad = \quad \begin{cases} f'(x) & \text{if } x \in X' \\ f(x) & \text{otherwise} \end{cases}$$

So $f[f']$ is the function f "updated" by f'. Also, if $f \in L^X$ and $X' \subseteq X$, then we right $f \lceil X'$ to denote the function in $L^{X'}$ defined by

$$f \lceil X'(x) = f(x) \text{ if } x \in X'.$$

Finally, if $X = \{x_1, \ldots, x_n\}$ and $\{l_1, \ldots, l_n\} \subseteq L$ then we write

$$\{x_1 \mapsto l_1, \ldots, x_n \mapsto l_n\}$$

to represent the function that maps each x_i to l_i.

Let $\langle L, \sqsubseteq \rangle$ be a lattice and X be a set. Then we can define a complete lattice on L^X as follows. If $f_1, f_2 \in L^X$, we abuse notation and write $f_1 \sqsubseteq f_2$ if for all $x \in X$, $f_1(x) \sqsubseteq f_2(x)$. Then $\langle L^X, \sqsubseteq \rangle$ forms a complete lattice, with the least upper bound and greatest lower bound operations for $F \subseteq L^X$ being given as follows.

$$\begin{aligned} (\sqcup F)(x) &= \sqcup \{ f(x) \mid f \in F \} \\ (\sqcap F)(x) &= \sqcap \{ f(x) \mid f \in F \} \end{aligned}$$

We also define $\bot(x) = \bot$ and $\top(x) = \top$.

Since $\langle L^X, \sqsubseteq \rangle$ is a complete lattice, monotonic functions $\overline{F} \in L^X \rightarrow L^X$ are guaranteed by Theorem 2.1 to have greatest and least fixpoints. We now give an alternative characterization of these fixpoints that will be useful later in the paper. We first introduce the following notation. Let $\overline{F} \in L^X \rightarrow L^X$, and let $g \in L^{X'}$, where $X' \subseteq X$. Then $\overline{F}_g$ is defined as

$$\overline{F}_g(f) = (\overline{F}(f[g]))[g].$$

Intuitively, $\overline{F}_g(f)$ "fixes" the values of $x \in X'$ to $g(x)$ and evaluates $\overline{F}$. Clearly, if $\overline{F}$ is monotonic then so is $\overline{F}_{\{x \mapsto l\}}$.

Now let $\overline{F} \in L^X \rightarrow L^X$ be monotonic, and let $X' \subseteq X$. We may now define the following functions.

$$\begin{aligned} \overline{F}_{\nu X'}(f) &= \overline{F}(\nu \overline{F}_{f \lceil X'}) \\ \overline{F}_{\mu X'}(f) &= \overline{F}(\mu \overline{F}_{f \lceil X'}) \end{aligned}$$

If $X' = \{x\}$ is a singleton set, then we sometimes right $\overline{F}_{\sigma x}$ in lieu of $\overline{F}_{\sigma \{x\}}$. Intuitively, $\nu(\mu)\overline{F}_{f \lceil X'}$ represents the greatest (least) fixpoint of $\overline{F}$ in which the values of $x \in X'$ are held fixed at $f(x)$. Note that $(\nu(\mu)\overline{F}_{f\lceil X'})\lceil X' = f\lceil X'$. So $\overline{F}_{\nu X'}(f)$ in essence ignores all the components of f except the value assigned to x, calculates the fixpoint of $\overline{F}$ holding x at the value assigned to it by f, and then applies $\overline{F}$ to the resulting mapping. Note that from the definition of $\overline{F}_g$, it follows that for all $x \not\in X'$, $\overline{F}_{\nu X'}(f)(x) = (\nu\overline{F}_{f\lceil X'})(x)$ and $\overline{F}_{\mu S'}(f)(x) = (\mu\overline{F}_{f\lceil X'})(x)$. It is also straightforward to establish that the monotonicity of $\overline{F}$ implies the monotonicity both of $\overline{F}_{\nu X'}$ and $\overline{F}_{\mu X'}$. We now have the following.

Theorem 2.2 *Let $\overline{F} \in L^X \rightarrow L^X$ be monotonic, with $X' \subseteq X$. Then the following are true.*

1. $\nu\overline{F} = \nu\overline{F}_{\nu X'}$
2. $\mu\overline{F} = \mu\overline{F}_{\mu X'}$

Proof. We only prove part 1; the proof of 2 is completely dual and left to the reader. On the basis of Theorem 2.1, we have the following.

$$\begin{array}{rcl} \nu\overline{F} & = & \sqcup\{\, f \in L^X \mid f \sqsubseteq \overline{F}(f)\,\} \\ \nu\overline{F}_{\nu X'} & = & \sqcup\{\, f \in L^X \mid f \sqsubseteq \overline{F}_{\nu X'}(f)\,\} \end{array}$$

We first show that $\nu\overline{F}_{\nu X'} \sqsubseteq \nu\overline{F}$, and we then establish that $\nu\overline{F} \sqsubseteq \nu\overline{F}_{\nu X'}$. To show the former, it suffices to establish that for every $f \sqsubseteq \overline{F}_{\nu X'}(f)$ there is a g such that $f \sqsubseteq g$ and $g \sqsubseteq \overline{F}(g)$. So fix such an f, and choose $g = \nu\overline{F}_{f\lceil X'}$. Let $X'' = X - \{x\}$. To show that $f \sqsubseteq g$, we first note that, $g\lceil X'' = \overline{F}_{\nu x}(f)\lceil X''$ and that $g\lceil X' = f\lceil X'$. Therefore, since $f \sqsubseteq \overline{F}_{\nu X'}(f)$, it follows that $f \sqsubseteq g$. We now must show that $g \sqsubseteq \overline{F}(g)$. Again, we have that $g\lceil X'' = \overline{F}(g)\lceil X''$, and since $g\lceil X' = f\lceil X'$ and $f \sqsubseteq \overline{F}_{\nu X'}(f) = \overline{F}(g)$, it follows that $g(x) \sqsubseteq \overline{F}(g)(x)$. Thus $g \sqsubseteq \overline{F}(g)$.

To establish that $\nu\overline{F} \sqsubseteq \nu\overline{F}_{\nu X'}$, it suffices to show that for any $g \sqsubseteq \overline{F}(g), g \sqsubseteq \overline{F}_{\nu X'}(g)$. So suppose that $g \sqsubseteq \overline{F}(g)$. It then follows that $\overline{F}(g) \sqsubseteq \overline{F}_{g\lceil X'}(g)$ and therefore that $g \sqsubseteq \nu\overline{F}_{g\lceil X'}$. Now the monotonicity of $\overline{F}$ implies that $\overline{F}(g) \sqsubseteq \overline{F}(\nu\overline{F}_{g\lceil X'}) = \overline{F}_{\nu X'}(g)$, and thus $g \sqsubseteq \overline{F}_{\nu X'}(g)$. □

We close this section by giving another alternative charactization of $\nu(\mu)\overline{F}$ in the special case that X is finite and $L = 2^S$, where S is a finite set. We first introduce some notation. Let $f_1, f_2 \in L^X$.

$$\begin{array}{rcl} (f_1 \cup f_2)(x) & = & f_1(x) \cup f_2(x) \\ (f_1 - f_2)(x) & = & f_1(x) - f_2(x) \end{array}$$

where if S_1 and S_2 are sets then $S_1 - S_2$ represents their difference. Also, in the lattice $\langle L^X, \subseteq\rangle$ we have that $\bot(x) = \emptyset$ and $\top(x) = S$ for all $x \in X$.

For $\overline{F} \in L^X \rightarrow L^X$, consider the functions C_ν and C_μ given as follows.

$$C_\nu(\overline{F}, f) = \textbf{if } \exists x.\, \exists s.\, s \in (f - \overline{F}(f))(x) \textbf{ then } C_\nu(\overline{F}, f - \bot[\{x \mapsto \{s\}\}]) \textbf{ else } f$$

$$C_\mu(\overline{F}, f) = \textbf{if } \exists x.\, \exists s.\, s \in (\overline{F}(f) - f)(x) \textbf{ then } C_\mu(\overline{F}, f \cup \bot[\{x \mapsto \{s\}\}]) \textbf{ else } f$$

Since S and X are finite, both functions are well-defined. We now have the following, which is an adaptation of a result given in [CP].

Theorem 2.3 *Let S and X be finite sets, with $L = 2^S$, and let $\overline{F} \in L^X \to L^X$ be monotonic. Then*

1. $\nu\overline{F} = C_\nu(\overline{F}, \top)$ *and*
2. $\mu\overline{F} = C_\mu(\overline{F}, \bot)$.

Proof. We only prove 1. Clearly, if $C_\nu(\overline{F}, g) = g$ then $g \subseteq \overline{F}(g)$, and this implies that $C_\nu(\overline{F}, g) \subseteq \nu\overline{F}$. To show that $\nu\overline{F} \subseteq C_\nu(\overline{F}, \top)$, it suffices to show that if $\nu\overline{F} \subseteq f$ and $s \in f - \overline{F}(x)$ for some x and s, then $\nu\overline{F} \subseteq f - \bot[\{x \mapsto \{s\}\}]$. So suppose that $\nu\overline{F} \subseteq f$ and $s \in f - \overline{F}(x)$. To establish that $\nu\overline{F} \subseteq f - \bot[\{x \mapsto \{s\}\}]$ we need only show that $s \notin (\nu\overline{F})(x)$. This follows from the fact that since $\overline{F}$ is monotonic, $\nu\overline{F} = \overline{F}(\nu\overline{F}) \subseteq \overline{F}(f)$ and $s \notin \overline{F}(f)(x)$. □

3 Syntax and Semantics of the μ-Calculus

This section introduces *labeled transition systems*, which are used as models for the μ-calculus. It then defines the syntax and semantics of the logic.

3.1 Transition Systems

Definition 3.1 *A* labeled transition system $\mathcal{T}$ *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 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.

3.2 Syntax of Formulas

The syntax of the modal μ-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. 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

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

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

Figure 1: The Semantics of Formulas.

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 $\mathcal{F}$ denote the set of all syntactically correct formulas.

Let $\Phi[X := \Gamma]$ represent the formula obtained by simultaneously substituting the formula Γ for the free occurrences of the variable X in Φ, with bound variables renamed as usual to prevent the capture of free variables in Γ. Then we may also define the following operators that are dual to the ones we have presented.

$$
\begin{array}{rcl}
\Phi_1 \vee \Phi_2 & = & \neg(\neg\Phi_1 \wedge \neg\Phi_2) \\
\langle a \rangle \Phi & = & \neg[a](\neg\Phi) \\
\mu X.\Phi & = & \neg\nu X.\neg(\Phi[X := \neg X])
\end{array}
$$

We may also generalize the substitution operator slightly as follows. Let $\Phi \in \mathcal{F}, \mathcal{X} \subseteq Var$, and $f \in \mathcal{F}^{\mathcal{X}}$. Then $\Phi[f]$ represents formula Φ with every free occurrence of $X \in \mathcal{X}$ replaced by $f(X)$, with bound variables in Φ renamed as necessary.

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. We also refer to a formula's maximal proper subformulas with a given property as the *top-level* subformulas having that property. We call a formula a ν-formula (μ-formula) if it has the form $\nu X.\Phi$ ($\mu X.\Phi$) for some X and Φ, and we let $|\Phi|$ denote the number of occurrences of operators and atomic formulas in Φ. We say that a formula is *closed* if it contains no occurrences of free variables and *simple* if it is of the form $\neg A$ for atomic A or contains only variables as proper subformulas. For example, $\neg A$ and $X_1 \wedge X_2$ are simple, while $\langle a \rangle (X_3 \vee X_4)$ is not; in the last case the formula contains a proper subformula— $X_3 \vee X_4$— that is not a variable.

3.3 Semantics of Formulas

The formal semantics of formulas appears in Figure 1. The definition is given relative 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 $e \in (2^{\mathcal{S}})^{Var}$ mapping variables to subsets of $\mathcal{S}$. Note that $e[\{X \mapsto S\}]$ is the environment resulting from "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 ensure that in $\nu X.\Phi$, the function defined by $\phi_e(S) = [\![\Phi]\!]e[\{X \mapsto S\}]$ is monotonic on the lattice of sets of states [C]. Accordingly, using Theorem 2.1 the semantics of $\nu X.\Phi$ can be given as the greatest fixpoint ϕ_e. In addition, for finite-state labeled transition systems the bodies of ν-formulas are continuous, and Kleene's Fixpoint Theorem [K] then provides the following iterative characterization of the semantics. Define ϕ_i by

$$\begin{aligned} \phi_0 &= \mathcal{S} \\ \phi_{i+1} &= [\![\Phi]\!]e[\{X \mapsto \phi_i\}] \text{ for } i \geq 1 \end{aligned}$$

Then $[\![\nu X.\Phi]\!]e = \cap_{i=0}^{\infty}\phi_i$. Formula $\mu X.\Phi$ can be characterized dually as $\cup_{i=0}^{\infty}\hat{\phi}_i$, where $\hat{\phi}_0 = \emptyset$ and $\hat{\phi}_{i+1} = [\![\Phi]\!]e[\{X \mapsto \hat{\phi}_i\}]$.

The semantics of a closed formula is not influenced by the initial environment, since all variables are bound by a fixpoint operator.

Lemma 3.2 *Let Φ be a closed formula, and e and e' be two environments. Then we have for every finite-state transition system and every valuation: $[\![\Phi]\!]e = [\![\Phi]\!]e'$.*

Thus we sometimes abuse notation and write $[\![\Phi]\!]$ to refer to the semantics of a closed formula Φ.

Finally, it is also possible to translate formulas into *positive normal form* (PNF), i.e. into formulas in which any variable is bound at most once and negation is only applied to atomic formulas and free variables. This is a consequence of the following lemma.

Lemma 3.3 *Let Φ be a formula in the modal μ-calculus. Then Φ can be translated into a formula Φ' in the logic extended with $\vee$, $\langle a\rangle$ and μ in $O(|\Phi|)$ time such that the following hold.*

- *The only negated subformulas of Φ' have the form $\neg A$, where A is atomic, or $\neg X$, where X is a free variable in Φ'.*
- *Any variable is bound at most once in Φ'.*
- *No free variable in Φ has the same name as any bound variable in Φ.*
- *$[\![\Phi]\!]e = [\![\Phi']\!]e$ for any e.*

The translation is done by "driving" negations inside the subformulas using dualities and renaming variables as appropriate. The negation property enjoyed by Φ' results from the syntactic restriction imposed on the bodies of fixpoint formulas and the duality of $\vee$ and $\wedge$, $\langle a\rangle$ and $[a]$, and μ and ν. Note that if Φ is closed, then the only negated subformulas of Φ' have the form $\neg A$. Moreover, it is also the case that $|\Phi'| \leq 2 * |\Phi|$.

In a PNF formula Φ, we often refer to a bound variable as a ν-variable if it is bound by ν and a μ-variable if it is bound by μ.

We close this section by stating some semantic equalities that hold on μ-calculus formulas.

Lemma 3.4 *Let $\Phi \in \mathcal{F}, X \in Var, e$ be an environment, and $\sigma \in \{\mu, \nu\}$.*

1. *If X is not free in Φ then $[\![\sigma X.\Phi]\!]e = [\![\Phi]\!]e$.*
2. *Let $Y \in Var$ not appear free in $\sigma X.\Phi$. Then $[\![\sigma X.\Phi]\!]e = [\![\sigma Y.(\Phi[X := Y])]\!]e$.*
3. *Let $\Gamma \in \mathcal{F}$. Then $[\![\Phi[X := \Gamma]]\!]e = [\![\Phi]\!]e[\{X \mapsto [\![\Gamma]\!]e\}]$.*
4. *$[\![\sigma X.\Phi]\!]e = [\![\Phi[X := \sigma X.\Phi]]\!]e$.*

Proof. Parts 1 and 2 follow by easy inductions on Φ. Parts 3 and 4 are restatements of Lemma 2.4 and Corollary 2.5 of [C]. □

Let $\mathcal{X} \subseteq Var$, with $f \in \mathcal{F}^{\mathcal{X}}$. Then we abuse notation and write $[\![f]\!]e$ to represent the function defined by

$$([\![f]\!]e)(X) = [\![f(X)]\!]e$$

for $X \in \mathcal{X}$. The following corollary is an immediate consequence of the previous lemma.

Corollary 3.5 *Let $\Phi \in \mathcal{F}, \mathcal{X} \subseteq Var$, and $f \in \mathcal{F}^{\mathcal{X}}$. Then $[\![\Phi[f]]\!]e = [\![\Phi]\!]e[[\![f]\!]e]$.*

3.4 Alternation Depth of Formulas

The complexity of model-checking algorithms is commonly expressed in terms of the *alternation depth* of the investigated formula [EL]. Intuitively, the alternation depth of a formula is the length of a maximal "chain" of nested non-closed fixpoint subformulas with alternating μ and ν fixpoint operators.

Definition 3.6 (Alternation Depth of Formulas)
Let Φ be a μ-calculus formula in PNF. Then the alternation depth, *$ad(\Phi)$, of Φ is defined inductively as follows.*

- *If Φ contains top-level closed 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 top-level closed fixpoint subformulas then $ad(\Phi)$ is defined as follows.*
  - *$ad(A) = ad(X) = 0$, for any atomic proposition A and variable X.*
  - *$ad(\neg A) = ad(\neg X) = 0$, for any atomic proposition A and free 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'.

If Φ is not in PNF, then $ad(\Phi) = ad(\Phi')$, where Φ' is the PNF formula equivalent to Φ obtained as a result of Lemma 3.3.

Example 3.7 For the formulas

$$\begin{aligned}
\Phi_1 &= \nu Z.\mu Y.[a]((A \wedge Z) \vee Y)\\
\Phi_2 &= \nu X_1.\mu X_2.(X_1 \vee X_2 \vee \nu Y_1.\mu Y_2.\nu Y_3.(Y_1 \wedge Y_2 \wedge Y_3))\\
\Phi_3 &= \mu X.(A \wedge \nu Y.(X \vee B))\\
\Phi_4 &= \nu Y.(Y \wedge \mu Z.(Z \vee X))\\
\Phi_5 &= \nu X.\mu Y.(Y \wedge \nu Z.(Z \wedge \mu W.(W \wedge X)))
\end{aligned}$$

we obtain $ad(\Phi_1) = 2$, $ad(\Phi_2) = 3$, $ad(\Phi_3) = 2$, $ad(\Phi_4) = 2$, and $ad(\Phi_5) = 4$.

The previous example illustrates the results implied by the definition of ad for different formulas. In the cases of Φ_3, Φ_4 and Φ_5, it also points out slight problems in the way the definition handles free variables. For example, $ad(\Phi_3) = 2$; however, since the variable Y does not appear free in the body of subformula $\nu Y.(X \vee B)$, there is no mutual dependency between this subformula and Φ_3. Also, $ad(\Phi_4) = 2$; however, the variable Y does not appear free within the subformula $\mu Z.(Z \vee X)$, and thus one might reasonably infer that this subformula is independent of its context and that no mutual dependency exists. Accordingly, the next section introduces an improved measure designed to overcome these shortcomings.

Reduced Alternation Depth of Formulas

The notion of *reduced alternation depth* of a formula is very similar to the one of alternation depth; the only difference concerns its treatment of free variables. We first introduce the following auxiliary definitions.

Definition 3.8 *Let Φ be a PNF formula. Then Φ' is* relatively closed *with respect to Φ if $FV(\Phi') \subseteq FV(\Phi)$ where $FV(\Psi)$ denotes the set of free variables in formula Ψ.*

For instance, consider

$$\Phi_4 = \nu Y.(Y \wedge \mu Z.(Z \vee X))$$

as presented in Example 3.7. $\mu Z.(Z \vee X)$ is not closed, but it is relatively closed with respect to Φ_4, since both formulas have the same set— $\{X\}$— of free variables.

The next definition characterizes a class of formulas having no "redundant" fixpoint operators. For the purposes of this discussion, if the variable X does not appear free Φ, then the σX is redundant in $\sigma X.\Phi$, since it does not influence the meaning of the formula.

Definition 3.9 *Let Φ be a PNF formula. Then Φ is in* fixpoint normal form *(FNF) if for every subformula $\sigma X.\Phi'$ of Φ, X appears free in Φ'.*

It turns out that every formula in PNF may be transformed into a formula in FNF, as the following lemma indicates.

Lemma 3.10 *Let Φ be a formula in PNF. Then there exists a formula Φ' in FNF such that $|\Phi'| \leq |\Phi|$ and $[\![\Phi']\!]e = [\![\Phi]\!]e$ for any environment e.*

The transformation used in the proof of the lemma proceeds by replacing every subformula of the form $\sigma X.\Gamma$ in which X does not appear free in Γ by Γ.

Reduced alternation depth may now be given as follows. The definition is very similar to the one of alternation depth.

Definition 3.11 (Reduced Alternation Depth of Formulas)
Let Φ be a μ-calculus formula in FNF. Then the reduced alternation depth*, $rad(\Phi)$, of Φ is defined inductively as follows.*

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

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

  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 top-level* relatively closed *fixpoint subformulas then $rad(\Phi)$ is defined as follows.*
  - *$rad(A) = rad(X) = 0$, for any atomic proposition A and variable X.*
  - *$rad(\neg A) = rad(\neg X) = 0$, for any atomic proposition A and free variable X.*
  - *$rad(\Phi_1 \wedge \Phi_2) = rad(\Phi_1 \vee \Phi_2) = max(rad(\Phi_1), rad(\Phi_2))$.*
  - *$rad([a]\Phi') = rad(\langle a \rangle \Phi') = rad(\Phi')$, for any action a.*
  - *Let $\sigma \in \{\mu, \nu\}$, and let $\overline{\sigma}$ be the dual of σ. Then*

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

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

If Φ is not in FNF, then $rad(\Phi) = rad(\Phi')$, where Φ' is the FNF formula equivalent to Φ obtained as a result of Lemmas 3.3 and 3.10.

To see the differences between alternation depth and reduced alternation depth, consider the formulas in Example 3.7. In the case of

$$\Phi_3 = \mu X.(A \wedge \nu Y.(X \vee B)),$$

since Y does not appear free in $X \vee B$, Φ_3 is not in FNF. The equivalent FNF formula is

$$\Phi_3' = \mu X.(A \wedge (X \vee B)),$$

and we obtain $rad(\Phi_3) = rad(\Phi_3') = 1 < 2 = ad(\Phi_3)$. In the case of

$$\Phi_4 = \nu Y.(Y \wedge \mu Z.(Z \vee X)),$$

Φ_4 is in FNF. Since $\mu Z.(Z \vee X)$ is relatively closed with respect to Φ_4, it follows that $rad(\Phi_4) = 1 < 2 = ad(\Phi_4)$. Formula Φ_5 is also in FNF, and the difference between $rad(\Phi_5)$ and $ad(\Phi_5)$ is even more pronounced, as $rad(\Phi_5) = 2 < 4 = ad(\Phi_5)$. Note that for Φ_1 and Φ_2 the notions of *ad* and *mad* coincide, as both formulas are in FNF and the closed fixpoint subformulas are identical with the relatively closed fixpoint subformulas. We often refer to a formula Φ as *alternation-free* when $rad(\Phi) \leq 1$.

The next theorem follows in a straightforward fashion from the structural similarity of the definitions of alternation depth and reduced alternation depth.

Theorem 3.12 *For any μ-calculus formula Φ, $rad(\Phi) \leq ad(\Phi)$.*

An Alternative Characterization

We now give an alternative characterization of reduced alternation depth for FNF formulas. The account explicitly quantifies the number of "alternations" within a formula and will be useful later in the paper.

Recall that since each FNF formula is also in PNF, each subformula of the form $\mu X.\Phi$ binds a unique variable. We can therefore define a relation on the bound variables in a FNF formula that indicates when a variable appears free in the scope of another.

Definition 3.13 *Let* Φ *be a FNF formula, and let* $\sigma_1, \sigma_2 \in \{\nu, \mu\}$*. Then* $X \; \mathit{afi}_\Phi \; Y$ *if* $\sigma_1 X.\Phi'$ *is a subformula of* Φ*,* $\sigma_2 Y.\Phi''$ *is a subformula of* Φ'*, and* X *appears free in* Φ''*.*

For example, in formula Φ_1 above we have that $Z \; \mathit{afi}_{\Phi_1} \; Y$. Note that if $\sigma Y.\Phi'$ is a relatively closed subformula of Φ then there can be no X such that $X \; \mathit{afi}_\Phi \; Y$.

Let Φ be a FNF formula. We say that a sequence $\langle X_1, \ldots, X_n \rangle$ of bound variables in Φ is an afi_Φ*-chain* if $X_i \; \mathit{afi}_\Phi \; X_{i+1}$ for $1 \leq i < n$. The following function defines the *alternation length* of a given afi_Φ-chain, where :: represents sequence concatenation.

Definition 3.14 *Let* Φ *be a FNF formula, and let* c *be an* afi_Φ*-chain in* Φ*. Then the* alternation length*,* $al(c)$*, of* c *is defined inductively as follows.*

$$al(c) = \begin{cases} 0 & \text{if } c = \langle\rangle \\ 1 & \text{if } c = \langle X \rangle \\ al(\langle X_2 \rangle :: c') & \text{if } c = \langle X_1, X_2 \rangle :: c' \text{ and } X_1, X_2 \text{ are either both } \nu\text{- or both} \\ & \mu\text{-variables} \\ 1 + al(\langle X_2 \rangle :: c') & \text{otherwise} \end{cases}$$

We now define our alternative characterization of *rad* as follows.

$$\mathit{rad}'(\Phi) = \mathit{max}\{\, al(c) \mid c \text{ is an } \mathit{afi}_\Phi\text{-chain in } \Phi \,\}$$

The following theorem establishes that this characterization is faithful.

Theorem 3.15 (Alternative Characterization of Reduced Alternation Depth)

Let Φ *be a FNF formula. Then* $\mathit{rad}(\Phi) = \mathit{rad}'(\Phi)$*.*

Proof. The proof proceeds by induction on $|\Phi|$. For the basis step we must prove the result for $|\Phi| = 1$. In this case Φ is an atomic proposition or a variable. In either case, $\mathit{rad}(\Phi) = \mathit{rad}'(\Phi) = 0$, since Φ contains no fixpoint subformulas and hence no bound variables.

For the induction step, fix $n > 1$. The induction hypothesis states that $\mathit{rad}(\Phi) = \mathit{mad}'(\Phi)$ for any Φ with $|\Phi| < n$; we must prove the result for Φ with $|\Phi| = n$. So fix Φ such that $|\Phi| = n$. We consider two cases.

- Φ contains top-level relatively closed fixpoint subformulas $\Gamma_1, \ldots, \Gamma_m$. In this case, $\mathit{rad}(\Phi) = \mathit{max}(\mathit{rad}(\Phi'), \mathit{rad}(\Gamma_1), \ldots, \mathit{rad}(\Gamma_m))$, where Φ' is obtained from Φ by replacing the Γ_i by new atomic propositions A_i. It follows that $|\Phi'| < |\Phi|$ and that $|\Gamma_i| < |\Phi|$ for each i. Thus the induction hypothesis guarantees that $\mathit{rad}(\Phi') = \mathit{rad}'(\Phi')$ and that $\mathit{rad}(\Gamma_i) = \mathit{rad}'(\Gamma_i)$ for each i. So it suffices to establish that $\mathit{rad}'(\Phi) = \mathit{max}(\mathit{rad}'(\Phi'), \mathit{rad}'(\Gamma_1), \ldots, \mathit{rad}'(\Gamma_n))$ in order to prove that $\mathit{rad}(\Phi) =$

$rad'(\Phi)$. But this follows from the fact that since each Γ_i is relatively closed with respect to Φ, any afi_Φ-sequence $\langle X_1, \ldots, X_k \rangle$ must include variables that are all bound either in the same Γ_i or in Φ'.

- Φ contains no top-level relatively closed fixpoint subformulas. The proof proceeds by a case analysis on the structure of Φ. Most cases are routine; we concentrate on the situation when Φ is of the form $\sigma X.\Phi'$. Since Φ is in FNF, it follows that X must appear free in Φ'. Thus we have

  $$rad(\Phi) = max(1, rad(\Phi'), 1 + rad(\overline{\sigma} X_1.\Phi_1), \ldots, 1 + rad(\overline{\sigma} X_m.\Phi_m)),$$

  where the $\overline{\sigma} X_i.\Phi_i$ are the top-level $\overline{\sigma}$-subformulas of Φ.

  Since $|\Phi'| < |\Phi|$ and $|\overline{\sigma} X_i.\Phi_i| < |\Phi|$ for each i, the induction hypothesis ensures that $rad(\Phi') = rad'(\Phi')$ and that $rad(\overline{\sigma} X_i.\Phi_i) = rad'(\overline{\sigma} X_i.\Phi_i)$ for each i. Thus, it is sufficient to show that

  $$rad'(\Phi) = max(1, rad'(\Phi'), 1 + rad'(\overline{\sigma} X_1.\Phi_1), \ldots, 1 + rad'(\overline{\sigma} X_m.\Phi_m))$$

  in order to show that $rad'(\Phi) = rad(\Phi)$. Consider a maximal afi_Φ-chain c of Φ such that $al(c) = rad'(\Phi)$. It can have one of three forms.

  - c contains only X. Then $rad'(\Phi) = 1$. In this case, Φ' can contain no fixpoint subformulas, since if it did they would have to be relatively closed. This implies that $rad(\Phi) = 1$ as well.
  - c has the form $\langle X, Y \rangle :: c'$, where Y is also bound by a σ. In this case, $\langle Y \rangle :: c'$ is a maximal $afi_{\Phi'}$ chain in Φ' with $rad'(\Phi') = al(\langle Y \rangle :: c') = al(c)$, so in this case $rad'(\Phi) = rad'(\Phi')$.
  - c has the form $\langle X, Y \rangle :: c'$, where Y is bound by a $\overline{\sigma}$. In this case, Y must be one of the X_i, where $\overline{\sigma} X_i.\Phi_i$ is a top-level $\overline{\sigma}$-subformula of Φ. It then follows that $\langle X_i \rangle :: c'$ is a maximal $afi_{\overline{\sigma} X_i.\Phi_i}$-chain with $rad(\overline{\sigma} X_i.\Phi_i) = al(\langle Y \rangle :: c') = al(c) - 1$, so here $rad'(\Phi) = 1 + rad'(\overline{\sigma} X_i.\Phi_i)$.

  Thus it follows that

  $$rad'(\Phi) = max(1, rad'(\Phi'), 1 + rad'(\overline{\sigma} X_1.\Phi_1), \ldots, 1 + rad'(\overline{\sigma} X_m.\Phi_m))$$

$\square$

4 Equational Systems

In order to facilitate the saving and reuse of intermediate results, our model-checking algorithm works on *equational* representations of μ-calculus formulas. This section presents the syntax and semantics of equational systems and defines the reduced alternation depth of systems; this notion turns out to be closely related to the analogous one defined in the previous section.

4.1 Syntax of Equational Systems

Formulas may be represented using systems of mutually recursive equations of the following form,[1] where each X_i is a distinct variable and each Φ_i is simple.

$$\begin{bmatrix} X_1 & \triangleleft\!\triangleright_1 & \Phi_1 \\ & \vdots & \\ X_n & \triangleleft\!\triangleright_n & \Phi_n \end{bmatrix} \qquad\qquad \triangleleft\!\triangleright \in \{\rightarrow, \leftarrow\}$$

Intuitively, the equation $X_i \rightarrow \Phi_i$ represents a *greatest fixpoint*, while $X_i \leftarrow \Phi_i$ represents a *least fixpoint*. In what follows we refer to $X_i \rightarrow \Phi_i$ as a *max equation*; in this case we say that X_i is a *max variable* (or has *parity* max). Similarly, $X_j \leftarrow \Phi_j$ is a *min equation* with *min variable* X_j (so X_j has *parity* min). A variable Y is *free* in E if it occurs in some right-hand side in E but is not a left-hand side. An equational system E is *closed* if it contains no free variables, i.e. if all variables in a right-hand side of some equation also appear as left-hand sides in E.

We place an additional restriction on the use of negation in the Φ_i for the purposes in this paper; $\neg X_j$ can be a right-hand side only if X_j is free 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 4.1 The following are equational systems whose correspondence to the formulas given in Example 3.7 will be established in Section 4.3.

$$E_1 = \begin{bmatrix} X_1 & \rightarrow & X_2 \\ X_2 & \leftarrow & [a]X_3 \\ X_3 & \leftarrow & X_4 \vee X_2 \\ X_4 & \leftarrow & X_5 \wedge X_1 \\ X_5 & \leftarrow & A \end{bmatrix} \qquad E_2 = \begin{bmatrix} X_1 & \rightarrow & X_2 \\ X_2 & \leftarrow & X_1 \vee X_3 \\ X_3 & \leftarrow & X_2 \vee X_4 \\ X_4 & \rightarrow & X_5 \\ X_5 & \leftarrow & X_6 \\ X_6 & \rightarrow & X_4 \wedge X_7 \\ X_7 & \rightarrow & X_5 \wedge X_6 \end{bmatrix}$$

$$E_3 = \begin{bmatrix} X_1 & \leftarrow & X_2 \wedge X_3 \\ X_2 & \leftarrow & A \\ X_3 & \rightarrow & X_1 \vee X_4 \\ X_4 & \rightarrow & B \end{bmatrix} \qquad E_4 = \begin{bmatrix} X_1 & \rightarrow & X_1 \wedge X_2 \\ X_2 & \leftarrow & X_2 \vee X_3 \end{bmatrix}$$

$$E_5 = \begin{bmatrix} X_1 & \rightarrow & X_2 \\ X_2 & \leftarrow & X_2 \wedge X_3 \\ X_3 & \rightarrow & X_3 \wedge X_4 \\ X_4 & \leftarrow & X_4 \wedge X_1 \end{bmatrix}$$

4.2 Semantics of Equational Systems

We use a two-step procedure to define the semantics of equational systems as mappings from left-hand side variables to sets of states. In the first step, we introduce a translation from equational systems to mappings of left-hand side variables to formulas containing no

---

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

free occurrences of left-hand side variables. The second step then composes the semantic function for formulas with this mapping to obtain the denotation of the system.

Let $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n \rangle$ be an equational system, with $X = \{X_1, \ldots, X_n\}$. The mappings $\Delta_{n+1}, \ldots, \Delta_1 \in \mathcal{F}^X$ may now be defined as follows.

- $\Delta^{n+1}(X_i) = \Phi_i$ for $1 \leq i \leq n$.
- For $1 \leq j \leq n$,

  $$\Delta^j(X_i) \;=\; \begin{cases} F(X_i, \diamond_i, \Delta^{j+1}(X_i)) & \text{if } X_i \equiv X_j \\ \Delta^{j+1}(X_i)[X_j := F(X_j, \diamond_j, \Delta^{j+1}(X_j))] & \text{otherwise} \end{cases}$$

  where

  $$F(X, \diamond, \Phi) \;=\; \begin{cases} \nu X.\Phi & \text{if } \diamond = \rightarrow \\ \mu X.\Phi & \text{otherwise} \end{cases}$$

Let $\Delta(E) = \Delta_1$. We now have the following.

Lemma 4.2 *Let $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n \rangle$ be an equational system. Then for any X_i, X_j with $1 \leq i, j \leq n$, X_i does not appear free in $\Delta(E)(X_j)$.*

Proof. The result follows from the fact that in $\Delta^k(X_j)$, $X_k, \ldots, X_n$ cannot appear free. $\square$

As a corollary, we have that if an equational system E is *closed*, then each formula $\Delta(E)(X_i)$ is also closed. We now define the semantics of equational systems as follows. Recall that if $f \in \mathcal{F}^{\mathcal{X}}$ for $\mathcal{X} \subseteq \mathit{Var}$, then $[\![f]\!]e$ is a function defined by $([\![f]\!]e)(X) = [\![f(X)]\!]e$ for $X \in \mathcal{X}$.

Definition 4.3 (Semantics of Equational Systems)
Let $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n \rangle$ be an equational system, $\mathcal{X} = \{X_1, \ldots, X_n\}$, and $e \in \mathit{Var} \rightarrow 2^{\mathcal{S}}$, where $\mathcal{S}$ is the set of states in a given labeled transition system. Then $[\![E]\!]e \in (2^{\mathcal{S}})^{\mathcal{X}}$ is defined by $([\![E]\!]e) = [\![\Delta(E)]\!]e$.

As with formulas, when E is closed then $[\![E]\!]e = [\![E]\!]e'$ for any two environments e and e'. When this is the case we write $[\![E]\!]$ to represent the (environment-independent) meaning of E.

Given an equational system $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n \rangle$ and an environment e, we write $e[E]$ to refer to the environment in which each X_i is updated by its value in $[\![E]\!]e$; that is, $e[E] = e[[\![E]\!]e]$, where $e[[\![E]\!]e]$ is as defined in Section 2.

We now give alternative characterizations for $[\![E]\!]e$, where E has a particular form. These alternative characterizations will be useful later when we prove the correctness of the model-checking algorithm. We first define the following notation.

Definition 4.4 *Let $\mathcal{S}$ be the set of states in a given labeled transition system, $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n \rangle$ be an equational system, with $\mathcal{X} = \{X_1, \ldots, X_n\}$, and e be an environment. Then $\overline{F}_e^E \in (2^S)^{\mathcal{X}} \rightarrow (2^S)^{\mathcal{X}}$ is defined as follows.*

$$\overline{F}_e^E(f) = \{X_1 \mapsto [\![\Phi_1]\!]e[f], \ldots, X_n \mapsto [\![\Phi_n]\!]e[f]\}$$

.

The following lemmas establish that $\llbracket E \rrbracket e$ is a fixpoint of $\overline{F}^E_e$.

Lemma 4.5 *Let* $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$ *be an equational system. Then* $\llbracket \Phi_1 \rrbracket e[E] = \llbracket \Delta(E)(X_1) \rrbracket e$.

Proof. Let $E_2 = \langle X_2 \lhd\!\rhd_2 \Phi_2, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$, and define σ as ν if $\lhd\!\rhd_1 = \longrightarrow$ and μ otherwise. Then we have the following.

$$\begin{array}{rcl}
\llbracket \Phi_1 \rrbracket e[E] & = & \llbracket \Phi[\Delta(E)] \rrbracket e \text{ by Corollary 3.5 and the definition of } e[E] \\
& = & \llbracket \Phi[\Delta(E_2)][X_1 := \sigma X_1.\Phi[\Delta(E_2)]] \rrbracket e \text{ by definition of } \Delta \\
& = & \llbracket \sigma X_1.\Phi[\Delta(E_2)] \rrbracket e \text{ by Lemma 3.4(4)} \\
& = & \llbracket \Delta(E)(X_1) \rrbracket e \text{ by definition of } \Delta
\end{array}$$

□

Lemma 4.6 *Let* $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$ *be an equational system. Then* $\llbracket \Phi_i \rrbracket e[E] = \llbracket \Delta(E)(X_i) \rrbracket e$ *for any* $1 \leq i \leq n$.

Proof. If $i = 1$ then the result follows immediately from Lemma 4.5. So let $i \geq 2$ and define $E_i = \langle X_i \lhd\!\rhd_i \Phi_i, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$. By Lemma 4.5, we have that

$$\llbracket \Phi_i \rrbracket e[E_i] = \llbracket \Delta(E_i)(X_i) \rrbracket e.$$

Since $X_i, \ldots, X_n$ cannot appear free in $\Delta(E_i)(X_j)$ for any $i \leq j \leq n$, it also follows from Corollary 3.5 and the definition of $e[E]$ that $e[E][E_i] = e[E]$. We now have the following.

$$\begin{array}{rcl}
\llbracket \Phi_i \rrbracket e[E] & = & \llbracket \Phi_i \rrbracket e[E][E_i] \\
& = & \llbracket \Delta(E_i)(X_i) \rrbracket e[E] \\
& = & \llbracket \Delta(E_i)(X_i)[\Delta(E)] \rrbracket e \text{ by Corollary 3.5} \\
& = & \llbracket \Delta(E)(X_i) \rrbracket e \text{ since } X_i, \ldots, X_n \text{ are not free in } \Delta(E_i)(X_i)
\end{array}$$

□

As an immediate corollary, we have the following.

Corollary 4.7 *Let* $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$ *be an equational system. Then* $\llbracket E \rrbracket e = \overline{F}^E_e(\llbracket E \rrbracket e)$ *for any* e.

Proof. Follows from Lemma 4.6 and the definition of $\overline{F}^E_e$. □

The next result establishes that, provided all the $\lhd\!\rhd_i$ coincide in an equational system E, $\llbracket E \rrbracket e$ may be identified with the appropriate extremal fixpoint of $\overline{F}$.

Theorem 4.8 *Let* $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$ *be an equational system and* e *be an environment.*

1. *If* $\lhd\!\rhd_i = \longrightarrow$ *for* $1 \leq i \leq n$ *then* $\llbracket E \rrbracket e = \nu \overline{F}^E_e$.

2. If $\lhd\!\rhd_i = \leftarrow$ *for* $1 \leq i \leq n$ *then* $[\![E]\!]e = \mu\overline{F}^E_e$.

Proof. We prove part 1 here; part 2 is similar and therefore omitted. We proceed by establishing that $[\![E]\!]e \subseteq \nu\overline{F}^E_e$ and $\nu\overline{F}^E_e \subseteq [\![E]\!]e$. For the former, it suffices to show that $[\![E]\!]e = \overline{F}^E_e([\![E]\!]e)$. But this follows from Corollary 4.7.

To show that $\nu\overline{F}^E_e \subseteq [\![E]\!]e$, we use induction on n, the number of equations in E.
BASIS: $n = 0$. The result follows trivially in this case.
INDUCTION: Fix $n \geq 0$. The induction hypothesis states that for any E having n equations, $\nu\overline{F}^E_e \subseteq [\![E]\!]e$ for any e. We must show the result for E with $n+1$ equations. So fix such an $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_{n+1} \lhd\!\rhd_{n+1} \Phi_{n+1}\rangle$, and let $E' = \langle X_2 \lhd\!\rhd_2 \Phi_2, \ldots, X_{n+1} \lhd\!\rhd_{n+1} \Phi_{n+1}\rangle$ and $\mathcal{X}' = \{X_2, \ldots, X_{n+1}\}$. The induction hypothesis guarantees that $\nu\overline{F}^{E'}_e \subseteq [\![E']\!]e$ for any e. On the basis of Theorem 2.2, to show that $\nu\overline{F}^E_e \subseteq [\![E]\!]e$ it suffices to establish that $\nu(\overline{F}^E_e)_{\nu X_1} \subseteq [\![E]\!]e$. This in turn follows from Theorem 2.1 if we can show that for any $f \subseteq ((\overline{F}^E_e)_{\nu X_1})(f)$, $f \subseteq [\![E]\!]e$. So fix such an f. From the definition of $(\overline{F}^E_e)_{\nu X_1}$ it follows that for any f,

$$\begin{aligned}
(\overline{F}^E_e)_{\nu X_1}(f)\lceil\mathcal{X}' &= \nu\overline{F}^{E'}_{e[\{X_1 \mapsto f(X_1)\}]} \\
&= [\![E']\!]e[\{X_1 \mapsto f(X_1)\}] \text{ by the induction hypothesis}
\end{aligned}$$

It therefore follows that:

$$\begin{aligned}
((\overline{F}^E_e)_{\nu X_1})(f)(X_1) &= [\![\Phi_1]\!]e[\{X_1 \mapsto f(X_1)\}][E'] \\
&= [\![\Phi_1[\Delta(E')]]\!]e[\{X_1 \mapsto f(X_1)\}] \text{ by Corollary 3.5}
\end{aligned}$$

Since $f(X_1) \subseteq ((\overline{F}^E_e)_{\nu X_1})(X_1)$, then, $f(X_1) \subseteq [\![\Phi_1[\Delta(E')]]\!]e[\{X_1 \mapsto f(X_1)\}]$. Thus $f(X_1) \subseteq [\![\nu X_1.(\Phi_1[\Delta(E')])]\!]e = ([\![E]\!]e)(X_1)$. Also, we have already established that $f\lceil\mathcal{X}' \subseteq ((\overline{F}^E_e)_{\nu X_1})(f)\lceil\mathcal{X}' = [\![E']\!]e[\{X_1 \mapsto f(X_1)\}]$; since $f(X_1) \subseteq ([\![E]\!]e)(X_1)$, monotonicity ensure that $[\![E']\!]e[\{X_1 \mapsto f(X_1)\}] \subseteq [\![E']\!]e[\{X_1 \mapsto ([\![E]\!]e)(X_1)\}] = ([\![E]\!]e)\lceil\mathcal{X}'$, and thus $f(X_i) \subseteq ([\![E]\!]e)(X_i)$ for $X_i \not\equiv X_1$. Thus $f \subseteq [\![E]\!]e$. □

4.3 Translating Formulas into Equational Systems

We now provide a connection between equational systems and the μ-calculus by defining a function *trans* that translates formulas into systems of equations. Given a μ-calculus formula Φ, the translation procedure first converts it into an equivalent FNF formula Φ'. It then builds an equational system that includes an equation for certain subformulas of Φ'. When a new equation is added to the equational system being generated, the parity of its left-hand-side variable is determined by the closest enclosing fixpoint operator. Appendix A contains a detailed description of *trans*; the equational systems in 4.1 are in fact the results of applying *trans* to the formulas in 3.7 after eliminating some superfluous equations with variable right-hand sides.

By judiciously implementing the renaming of bound variables that *trans* performs, the procedure can be made to investigate each subformula at most once of the FNF formula Φ' that it produces, and since $|\Phi'| \leq 2 * |\Phi|$ *trans* works in linear time. Moreover, the number of equations in the resulting simple equational system E does not exceed the size

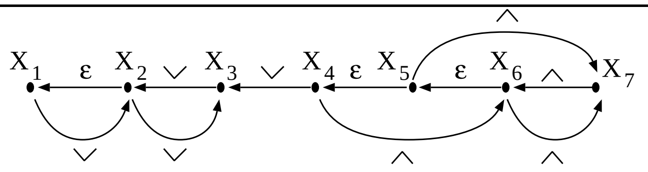

Figure 2: The Dependency Graph for Equational System E_2 in Example 4.1.

of the formula Φ, even though $|\Phi'|$ can exceed $|\Phi|$, because each subformula of Φ' is turned into at most one equation and negated subformulas of Φ' are turned into one equation. The equational systems produced by *trans* also preserve the semantics of the formulas given as inputs, as the next theorem demonstrates.

Theorem 4.9 *Let Φ be a formula with $X_1, \ldots$ not appearing free Φ, and let e be an environment. Then $[\![\Phi]\!]e = ([\![trans(\Phi)]\!]e)(X_1)$.*

Proof. See Appendix A. □

The model-checking algorithm presented in the next section computes $[\![E]\!]e$ for closed equational systems E and an environment e. Theorem 4.9 indicates that the algorithm may also be used (in conjunction with *trans*) as a model checker for the μ-calculus.

4.4 Equational Systems as Graphs

In this section we introduce a graph-based representation for equational systems that we use in later sections to guide the computation performed by the model-checking algorithm and to characterize the algorithm's complexity. Let E be an equational system $E = \langle X_1 \lhd\!\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\!\rhd_n \Phi_n \rangle$. Then the *dependency graph* G_E of E is a graph $\langle V, \rightarrow \rangle$, where the set V of vertices is $\{X_1, \ldots, X_n\}$ and the set $\rightarrow$ of labeled edges is defined as follows.

- $X_i \xrightarrow{\vee} X_j$ if for some X_k either $X_j \lhd\!\!\rhd_i X_i \vee X_k$ or $X_j \lhd\!\!\rhd_i X_k \vee X_i$ is an equation in E.
- $X_i \xrightarrow{\wedge} X_j$ if for some X_k either $X_j \lhd\!\!\rhd_i X_i \wedge X_k$ or $X_j \lhd\!\!\rhd_i X_k \wedge X_i$ is an equation in E.
- $X_i \xrightarrow{\langle a \rangle} X_j$ if $X_j \lhd\!\!\rhd_i \langle a \rangle X_i$ is in E.
- $X_i \xrightarrow{[a]} X_j$ if $X_j \lhd\!\!\rhd_i [a] X_i$ is in E.
- $X_i \xrightarrow{\varepsilon} X_j$ if $X_j \lhd\!\!\rhd_i X_i$ is in E.

In what follows, we write $X_i \dashrightarrow X_j$ if there is an edge in G_E from X_i to X_j and $X_i \dashrightarrow^* X_j$ if there is a path from X_i to X_j in G_E. Intuitively, $X_i \dashrightarrow X_j$ signifies that the meaning of X_i directly influences the meaning of X_j. As an example, the graph for the equational system E_2 in Example 4.1 appears in Figure 2. The next theorem characterizes the complexity of building G_E.

Theorem 4.10 *Let $E = \langle X_1 \lhd\!\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\!\rhd_n \Phi_n \rangle$ be an equational system. Then G_E contains n vertices and no more than $2n$ edges and may be constructed in $O(|E|)$ time.*

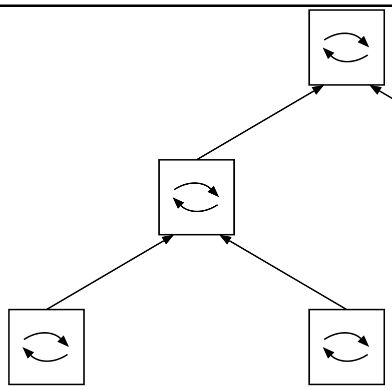

Figure 3: The Dependencies between and within Hierarchical Components.

Proof. Clearly, $G_E = \langle V, \rightarrow \rangle$ contains one vertex for each equation of E, and hence $|V| = n$. Also, from the definition of L and the fact that the right-hand sides in E are simple it follows that each vertex can have at most two incoming edges. Thus $|\rightarrow| \leq 2n$. Regarding the complexity of constructing G_E, consider the following technique for building G_E.

> For $1 \leq i \leq n$ the equations $X_i \lhd\!\rhd_i \Phi_i$ are processed one at a time, inserting an edge $X_j \overset{l}{\dashrightarrow} X_i$ for each variable X_j in Φ_i, with the label l determined by the form of Φ_i.

Obviously, this construction is linear in the size of E. □

We close this section by introducing some notation used later in the paper. If X_i and X_j are vertices in G_E, then we write $X_i \dashrightarrow_k X_j$ if $X_i \dashrightarrow X_j$ and $i, j \geq k$. Also, if C is a (not necessarily contiguous) sublist of E, then we refer to the subgraph of G_E induced by C as G_C, with $\rightarrow_C$ denoting the edges in G_C.

4.5 Hierarchical Components of Equational Systems

We now develop a graph-based characterization of certain of the subsystems of a given equational system E and show how the characterization may be used to determine $[\![E]\!]e$.

Let $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$ be an equational system. From the definition of G_E given in the previous section, it follows that if X_i and X_j are such that $X_i \dashrightarrow^* X_j$ and $X_j \dashrightarrow^* X_i$, then the semantics of X_i affects the semantics 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 in the meaning 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 the meaning of X_i is independent of the meaning of X_j. In this case we say that there is a *hierarchical* dependency from X_i to X_j.

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 only hierarchical dependencies can exist between two variables in distinct strongly connected components.

---

[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 $V_i, V_j \in \mathcal{V}$.

We may also define a total ordering on the strongly connected components of G_E as follows. First, let C_E be the *condensation graph* of G_E. C_E is the graph having the strongly connected components C_i of G_E as its vertices, with edges $C_i \rightarrow C_j$ if for vertices X in C_i and X' in C_j, $X \dashrightarrow X'$. Note that C_E is acyclic. Then we may topologically sort C_E, yielding the total order $C_1, \ldots, C_m$. Notice that if $C_i \rightarrow C_j$ then $i < j$.

In the topological ordering $C_1, \ldots, C_m$ of the strongly connected components of G_E, each left-hand-side variable X_j of E appears in exactly one of the C_i. Each C_i also gives rise to a subsystem of E as follows. Suppose C_i contains j variables. Then the associated subsystem consists of the j equations whose left-hand sides are in C_i, with the equations in the same relative order as they appeared in E. We often abuse terminology and use C_i to refer both to itself and to its associated subsystem in E, which we sometimes call a *hierarchical component* of E. It should be noted that the construction of $C_1, \ldots, C_m$ ensures that whenever X appears free in (a right-hand side of) C_j, then X is free in E or it is in some C_i for $i < j$ (i.e. C_i precedes C_j in the ordering).

As an example, consider E_2 in Example 4.1 with its dependency graph shown in Figure 2. This graph has two strongly connected components giving rise to the following hierarchical components.

$$C_1 = \begin{bmatrix} X_4 & \rightarrow & X_5 \\ X_5 & \leftarrow & X_6 \\ X_6 & \rightarrow & X_4 \wedge X_7 \\ X_7 & \rightarrow & X_5 \wedge X_6 \end{bmatrix} \qquad C_2 = \begin{bmatrix} X_1 & \rightarrow & X_2 \\ X_2 & \leftarrow & X_1 \vee X_3 \\ X_3 & \leftarrow & X_2 \vee X_4 \end{bmatrix}$$

The ordering for the hierarchical components is C_1, C_2, since $C_1 \rightarrow C_2$ is the only edge in C_E.

We now show that the semantics of E may be computed by determining the semantics of $C_1, \ldots, C_m$ in order. We first introduce the following auxiliary notion. Let e be an environment; we define e_i as follows.

$$e_i = \begin{cases} e & \text{if } i = 0 \\ e_{i-1}[C_i] & \text{if } 0 < i \leq m \end{cases}$$

Intuitively, e_i is the environment that results by processing each of the C_i in turn, beginning with C_1. Processing C_i involves determining its semantics in the environment resulting from the handling of C_{i-1} and assigning the results to the appropriate variables in the environment. The next theorem shows a relationship between $[\![C_i]\!]e_{i-1}$ and $[\![E]\!]e$.

Theorem 4.11 *Let E be an equational system with n equations and e be an environment.*

1. *The topological ordering $C_1, \ldots, C_m$ of the strongly connected components of G_E can be determined in $O(|E|)$ time.*

2. *If X_j is in C_i, then $([\![E]\!]e)(X_j) = ([\![C_i]\!]e_{i-1})(X_j)$.*

Proof.

1. To determine the hierarchical components of an equational system E we use the dependency graph G_E that can be constructed in $O(|E|)$ time (cf. Theorem 4.10). Based on G_E we construct the condensation graph in C_E in $O(|E|)$ time as the strongly connected components of G_E can be established $O(2n) = O(|E|)$ time. C_E may also be sorted into $C_1, \ldots, C_m$ in $O(2n) = O(|E|)$ time using depth-first search.

2. Given a directed acyclic graph $\langle V, \rightarrow \rangle$, we define the *height* of $v \in V$ as follows.

$$ht(v) = 1 + max\{\, ht(v') \mid v' \rightarrow v \,\}$$

where $max\ \emptyset = 0$.

The proof now proceeds by induction on $ht(C_i)$ in C_E. In the basis case, $ht(C_i) = 1$, which implies that the variables appearing free in C_i must also appear free in E. From the definition of Δ, then, it follows that if X_j is in C_i, then $\Delta(E)(X_j)$ and $\Delta(C_i)(X_j)$ are syntactically identical. Morever, for any variable Y appearing free in E, we have that $e(Y) = e_i(Y)$ for any e. Thus,

$$(\llbracket E \rrbracket e)(X_j) = \llbracket \Delta(E)(X_j) \rrbracket e = \llbracket \Delta(C_i)(X_j) \rrbracket e = (\llbracket C_i \rrbracket e_{i-1})(X_j).$$

For the induction step, fix $k > 1$. The induction hypothesis states that the result holds for any C' such that $ht(C') < k$; we must show the result for C_i where $ht(C_i) = k$. From the definition of the condensation graph C_E, it follows that the only variables that can appear free in C_i either are also free in E or appear as left-hand sides in hierarchical components C' with $ht(C') < ht(C_i)$. Suppose that X_j is in C_i. Then $\Delta(C_i)(X_j)$ is of the form $\sigma X_j.\Gamma$. Let $X_{l_1}, \ldots, X_{l_j}$ be the variables that are left-hand sides in E and that appear free in Γ. From the definition of Δ, it follows that $\Delta(E)(X_j)$ is syntactically identical to $\Gamma[X_{l_1} := \Delta(E)(X_{l_1}), \ldots, X_{l_j} := \Delta(E)(X_{l_j})]$, and thus that

$$(\llbracket E \rrbracket e)(X_j) = \llbracket \Delta(C_i)(X_j)[X_{l_1} := \Delta(E)(X_{l_1}), \ldots, X_{l_j} := \Delta(E)(X_{l_j})] \rrbracket e.$$

Thus it suffices to show that

$$\llbracket \Delta(C_i)(X_j) \rrbracket e_{i-1} = \llbracket \Delta(C_i)(X_j)[X_{l_1} := \Delta(E)(X_{l_1}), \ldots, X_{l_j} := \Delta(E)(X_{l_j})] \rrbracket e$$

Consider a particular X_{l_m}. From the definition of e_{i-1} and the fact that X_{l_m} belongs to a C' with $ht(C') < ht(C_i)$, it follows from the induction hypothesis that $e_{i-1}(X_{l_m}) = (\llbracket E \rrbracket e)(X_{l_m}) = \llbracket \Delta(E)(X_{l_m}) \rrbracket e$. But by Lemma 3.4(3), we have that $\llbracket \Delta(C_i)(X_j) \rrbracket e_{i-1} = \llbracket \Delta(C_i)(X_j)[X_{l_m} := \Delta(E)(X_{l_m})] \rrbracket e_{i-1}[\{X_{l_m} \mapsto e(X_{l_m})\}]$. By repeating this line of reasoning to each of the X_{l_m}, the desired result follows.

□

This theorem implies the following technique for computing $\llbracket E \rrbracket e$. First, compute the sequence $C_1, \ldots, C_m$ of hierarchical components; then, beginning with C_1, compute $\llbracket C_i \rrbracket$ in an environment containing the results of determining $\llbracket C_1 \rrbracket, \ldots, \llbracket C_{i-1} \rrbracket$.

4.6 Reduced Alternation Depth of Equational Systems

The previous section detailed an approach to computing $\llbracket E \rrbracket e$ that relied on computing the semantics of the hierarchical components C_l of E. In this section, we define a partitioning of equations within such a C_l that the model-checking algorithm uses to guide the meaning of C_l. Intuitively, equations within C_l are grouped according to their *nesting depth*, which is, given a left-hand-side variable X_i, a bound for the number of alternating parities that occurs on a cycle passing X_i and nodes with higher index than i in G_{C_l}

The *reduced alternation depth* of an equational system is then defined as the maximal nesting depth of any equation in the system. We show that this notion is consistent with the definition of reduced alternation depth for formulas given in Section 3.4.

Definition 4.12 (Nesting Depth of Equations)
Let $E = \langle X_1 \mathrel{\lhd\!\!\rhd}_1 \Phi_1, \ldots, X_n \mathrel{\lhd\!\!\rhd}_n \Phi_n \rangle$ *be an equational system with hierarchical components* $C_1, \ldots, C_m$. *Also let* $\sigma \in \{max, min\}$ *with* $\overline{\sigma}$ *its dual. Assume that* $X_i \in C_l$ *and that* X_i *has parity* σ. *Then the* nesting depth *of* X_i *is defined as follows.*

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

The nesting depth of C_l *is defined as*

$$nd(C_l) = max\{nd(X_i) | X_i \in C_l\}$$

In this definition, note that $X_j \dot{\rightarrow}_{C_l,i}{}^{*} X_i$ holds if and only if $j > i$ (so the equation for X_j occurs after the one for X_i in E) and there is a path in G_{C_l} from X_j to X_i that passes through vertices numbered higher than i.
A nesting level within a hierarchical component contains all equations having the same nesting depth.

Definition 4.13 (Nesting Levels)
Let $E = \langle X_1 \mathrel{\lhd\!\!\rhd}_1 \Phi_1, \ldots, X_n \mathrel{\lhd\!\!\rhd}_n \Phi_n \rangle$ *be an equational system with hierarchical component* C. *Then the set of variables in* C *may be partitioned into* nesting levels $E_1, \ldots, E_{nd(C_l)}$, *where* $E_i = \{\, X_j \mid nd(X_j) = i \,\}$.

Given two nesting levels E_i and E_j of C we call E_j *lower* than E_i if $j < i$ and *higher* if $j > i$. As was the case with hierarchical components, we often abuse terminology by referring to the equations whose left-hand sides are the variables in a nesting level as the nesting level itself.

Nesting levels may be further decomposed into hierarchically structured *blocks* of equations having the same parity as follows. Let E_l be a nesting level of a hierarchical component C, and let G_{E_l} be the subgraph of G_E induced by E_l. We may now compute the condensation graph C_{E_l} of G_{E_l}; let $B_{l,1}, \ldots, B_{l,b_l}$ be the vertices of C_{E_l} (and thus the strongly connected components of G_{E_l}), sorted topologically. From the fact that each variable in E_l has the same nesting depth, it follows that each of the $B_{l,k}$ may only contain either max-variables or min-variables, but not both. We sometimes refer to the sequence of blocks $\langle B_{l,1}, \ldots, B_{l,b_l} \rangle$ as the *block list* of E_l. Also, if $E_1, \ldots, E_k$ are the nesting levels of C (sorted in order of increasing nesting depth), and each E_i has block list $\mathcal{B}_i$ then we refer to $\mathcal{B}_1 :: \cdots :: \mathcal{B}_k$ as the block list of C. Note that the blocks in lower nesting levels appear before those of higher levels. Finally, if the topolically sorted hierarchical components of E are $C_1, \ldots, C_m$, and the block list of each C_i is $\mathcal{C}_i$, then $\mathcal{C}_1 :: \cdots :: \mathcal{C}_m$ is the block list of E. Intuitively, the blocks in the block list of E are sorted on the basis of the order in which they should be processed in order to compute $[\![E]\!]e$.

As an example, the equational system E_5 given in Example 4.1 consists of a single hierarchical component C having two nesting levels; E_2 holding the first equation and E_1 holding the last three equations. Nesting level E_1 holds three blocks, one for each equation.

4.6.1 Reduced Alternation Depth

The reduced alternation depth of an equational system is now defined as the maximal nesting depth of its hierarchical components.

Definition 4.14 (Reduced Alternation Depth of Equational Systems)
Let $E = \langle X_1 \Diamond_1 \Phi_1, \ldots, X_n \Diamond_n \Phi_n \rangle$ *be an equational system with hierarchical components* $C_1, \ldots, C_m$*. Then the* reduced alternation depth *of* E *is given by*

$$rad(E) = max\{nd(C_l) | 1 \leq l \leq m\}$$

When E is such that $rad(E) = 1$, then we say that E is *alternation-free.*

Example: Recall the equational systems given in Example 4.1. E_1 contains two hierarchical components, C_1 holding the last equation and C_2 holding the remaining equations . Therefore, we have $nd(X_5) = nd(C_1) = 1$, $nd(X_4) = nd(X_3) = nd(X_2) = 1$, $nd(X_1) = 2$, and $nd(C_2) = rad(E_1) = 2$. On the other hand, E_2 contains two hierarchical components, with C_1 containing X_4, X_5, X_6 and X_7 and C_2 comprising X_1, X_2 and X_3. We derive that $rad(E_2) = 3$ from the following observations.

- $nd(X_7) = nd(X_6) = 1$, $nd(X_5) = 2$, $nd(X_4) = 3$, so $nd(C_1) = 3$.
- $nd(X_3) = nd(X_2) = 1$ and $nd(X_1) = 2$, so $nd(C_2) = 2$.

Furthermore, $rad(E_3) = 1$, $rad(E_4) = 1$, and $rad(E_5) = 2$

The above definition clearly differs from the one given in Section 3.4; in that, for example, it is possible to have formulas with reduced alternation depth zero, while here every equational system has reduced alternation depth at least one. However, the notions are very closely related, as the next theorem indicates.

Theorem 4.15 *Let* Φ *be a FNF formula.*

1. *If* $rad(\Phi) = 0$ *then* $rad(trans(\Phi)) = 1$.
2. *If* $rad(\Phi) > 0$ *then* $rad(trans(\Phi)) = rad(\Phi)$.

Proof.

1. Suppose that $rad(\Phi) = 0$. From the definition of rad it follows that Φ must contain no fixpoint operators, and this implies that $trans(\Phi)$ contains only max equations. Therefore $rad(trans(\Phi)) = 1$.

2. Now suppose that $rad(\Phi) > 0$. The desired result is an immediate consequence of Theorem 3.15 and the following observations. Let X and Y be bound variables in Φ, with X_i and X_j the variables that $trans$ replaces them with in $trans(\Phi)$, respectively.

   - $X \; afi_{\Phi} \; Y$ if and only if X_i and X_j belong to the same strongly connected component in $G_{trans(\Phi)}$.
   - Suppose further that X_j and X_i both belong to strongly connected component C of $trans(\Phi)$. Then $X_j \dot{\rightarrow}_{C,i}{}^{*} X_i$ and $X_i \dot{\rightarrow}_{C,i}{}^{*} X_j$ if and only if there is an afi_{Φ}-chain $\langle X, \ldots, Y \rangle$.

□

We now characterize the complexity of determining the nesting depth and block information, and hence the reduced alternation depth information for an equational system E.

Theorem 4.16 *Given an equational system E, we have that*

1. *alternation-freedom can be established in $O(|E|)$ time, and*
2. *the nesting depth information can be determined in $O(|E|^2)$ time including the partitioning into blocks.*

Proof. Theorem 4.11 establishes that the hierarchical components of E may be computed in in $O(|E|)$ time.

1. By scanning the hierarchical components in linear time we may determine whether:
    - at least one of them contains both max and min equations (this implies that E is not alternation-free as there exist mutual dependencies between equations with distinct parities); or
    - none of them does, which implies that E is alternation-free.
2. Let C_l be a hierarchical component of E. To determine the nesting depth of an equation $X_i \triangleleft\!\triangleright_i \Phi_i$ in C_l we must determine all variables $X_j \in C_l$ such that $X_j \dashrightarrow^*_{C_l,i} X_i$ and $X_i \dashrightarrow^*_{C_l,i} X_j$ (cf. Definition 4.12). These variables and the maximum of their nesting depths can be determined in $O(|C_l|)$ time using a variant of Dijkstra's single-source algorithm. Performing this for every node leads to an $O(|C_l|^2)$ algorithm for determining the nesting depths of all equations in C_l. Thus the time needed to compute the nesting level information for all equations in E is $O(\Sigma_l |C_l|^2) = O(|E|^2)$. Having the nesting levels at hand, they can be partitioned into their blocks in linear time.

□

4.6.2 A Semantic Characterization

We close this section by proving an alternative characterization of $\llbracket E \rrbracket e$ that we use as the basis for the model-checking algorithm in the next section. We begin by establishing the following lemma.

Lemma 4.17 *Let $E = \langle X_1 \triangleleft\!\triangleright_1 \Phi_1, \ldots, X_n \triangleleft\!\triangleright_n \Phi_n \rangle$ be an equational system with block list $B_1, \ldots, B_k$, and let E_{k-1} represent the sublist of E with equations in B_k removed. Also let $\mathcal{X}_k$ be the set of left-hand side variables in B_k and define G_k^E as*

$$G_k^E(f) = \overline{F}^{B_k}_{e[\llbracket E_{k-1} \rrbracket e[f]]}(f).$$

1. *If B_k is a max block then $(\llbracket E \rrbracket e)\lceil \mathcal{X}_k = \nu G_k^E$.*
2. *If B_k is a min block then $(\llbracket E \rrbracket e)\lceil \mathcal{X}_k = \mu G_k^E$.*

Proof. We prove the first part; the second is dual. So suppose B_k has form $\langle X_{k_1} \rightarrow \Phi_{k_1}, X_{k_i} \rightarrow \Phi_{k_i} \rangle$. We now define block B'_k by

$$B'_k(X_{k_j}) = B_k(X_{k_j})[\Delta(E_{k-1})].$$

It is straightforward to show that $\Delta(B'_k) = \Delta(E)\lceil \mathcal{X}_k$ and thus that $\llbracket B'_k \rrbracket e = (\llbracket E \rrbracket e)\lceil \mathcal{X}_k$. Also, on the basis of Corollary 3.5 it is easy to show that for any f, $\overline{F}_e^{B'_k}(f) = G_k^E(f)$, whence $\nu \overline{F}_e^{B'_k} = \nu G_k^E$. But by Theorem 4.8 we know that $\nu \overline{F}_e^{B'_k} = \llbracket B'_k \rrbracket e$, so we have that $\nu G_k^E = \nu F_e^{B'_k} = \llbracket B'_k \rrbracket e = (\llbracket E \rrbracket e)\lceil \mathcal{X}_k$. □

Intuitively, $G_k^E(f)$ "assigns" the values in f to the left-hand side variables of B_k, evaluates the rest of E under this assignment, and then evaluates the right-hand sides of B_k using these values and those in f. This lemma thus shows that the extremal fixpoints of $G_k^E(f)$ may be used to characterize the meaning of B_k in the context of E.

The next result may be used to show that the semantics of E can be calculated in a "block-wise" fashion.

Theorem 4.18 *Let $E = \langle X_1 \lhd\!\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\!\rhd_n \Phi_n \rangle$ be an equational system with block list $B_1, \ldots, B_k$, and let E_{k-1} be the sublist of E with equations in B_k removed. Also define $G_{\nu,k}^E$ by*

$$G_{\nu,k}^E(f) = \nu \overline{F}^{B_k}_{e[\llbracket E_{k-1} \rrbracket e[f]]}[\llbracket E_{k-1} \rrbracket e[f]]$$

and $G_{\mu,k}^E$ by

$$G_{\nu,k}^E(f) = \mu \overline{F}^{B_k}_{e[\llbracket E_{k-1} \rrbracket e[f]]}[\llbracket E_{k-1} \rrbracket e[f]].$$

Then we have the following

1. *If B_k is a max block then $\llbracket E \rrbracket e = \nu G_{\nu,k}^E$.*
2. *If B_k is a min block then $\llbracket E \rrbracket e = \mu G_{\mu,k}^E$.*

Proof. We only prove the first part of the theorem; the proof of the second is completely dual. So assume that B_k is a max block. $G_{\nu,k}^E$ is clearly monotonic, and hence $\nu G_{\nu,k}^E$ is well-defined. Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ and $\mathcal{X}_k$ be the set of left-hand sides of X_k. The proof consists of two pieces.

- $\llbracket E \rrbracket e \subseteq \nu G_{\nu,k}^E$. Since $\nu G_{\nu,k}^E$ is the maximum fixpoint of $G_{\nu,k}^E$, it suffices to show that $\llbracket E \rrbracket e = G_{\nu,k}^E(\llbracket E \rrbracket e)$. We have the following.

$$\begin{aligned}
G_{\nu,k}^E(\llbracket E \rrbracket e) &= \nu \overline{F}^{B_k} e[\llbracket E_{k-1} \rrbracket e[\llbracket E \rrbracket e]][\llbracket E_{k-1} \rrbracket e[\llbracket E \rrbracket e]] && \text{by definition} \\
&= (\llbracket B_k \rrbracket e[(\llbracket E \rrbracket e)\lceil(\mathcal{X} - \mathcal{X}_1)])[\llbracket E_{k-1} \rrbracket e[E]] && \text{by Theorem 4.8 and the fact that no } X \in \mathcal{X}_1 \text{ appears free in } B_k \\
&= ((\llbracket E \rrbracket e)\lceil \mathcal{X}_1)[\llbracket E_{k-1} \rrbracket e[E]] \\
&= \llbracket E \rrbracket e
\end{aligned}$$

- $\nu G_{\nu,k}^E \subseteq \llbracket E \rrbracket e$. It suffices to show that for any $f \subseteq G_{\nu,k}^E(f)$, $f \subseteq \llbracket E \rrbracket e$. We first may characterize $\llbracket E \rrbracket e$ as follows.

$$\begin{aligned}
\llbracket E \rrbracket e &= (\llbracket B_k \rrbracket e[\llbracket E_2 \rrbracket e[E]])[\llbracket E_2 \rrbracket e[E]] \\
&= \nu \overline{F}^{B_k}_{e[\llbracket E_2 \rrbracket e[E]]}[\llbracket E_2 \rrbracket e[E]] && \text{by Theorem 4.8}
\end{aligned}$$

Now consider $f \subseteq G^E_{\nu,k}(f)$. It follows that $f\lceil\mathcal{X}_1 \subseteq \nu\overline{F}^{B_k}_{e[\llbracket E_2 \rrbracket e[f]]}$ and that $f\lceil(\mathcal{X}-\mathcal{X}_1) \subseteq \llbracket E_2 \rrbracket e[f]$. On the basis of the former, it follows that $f \subseteq \nu\overline{F}^{B_k,E}_e$, where

$$\overline{F}^{B_k,E}_e(g) = \overline{F}^{B_k}_{e[\llbracket E_{k-1} \rrbracket e[g]]}(g).$$

But on the basis of Lemma 4.17 we have that $\nu\overline{F}^{B_k,E}_e = \llbracket B_k \rrbracket e[E] = (\llbracket E \rrbracket e)\lceil\mathcal{X}_1$; thus, by monotonicity, it follows that $\llbracket E_2 \rrbracket e[f] \subseteq \llbracket E_2 \rrbracket e[E]$. So $f\lceil\mathcal{X}_1 \subseteq (\llbracket E \rrbracket e)\lceil\mathcal{X}_1$ and $f\lceil(\mathcal{X} - \mathcal{X}_1)$, whence $f \subseteq \llbracket E \rrbracket e$.

□

5 The Model-Checking Algorithm

This section presents the model-checking algorithm *solve* that, given a closed equational system $E = \langle X_1 \mathrel{\triangleleft\!\!\triangleright}_1 \Phi_1, \ldots, X_n \mathrel{\triangleleft\!\!\triangleright}_n \Phi_n \rangle$ and a labeled transition system $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle$, computes $\llbracket E \rrbracket$ with respect to a valuation $\mathcal{V}$. We begin by giving an overview of *solve*; succeeding sections then describe the major components of the algorithm in more detail.

As with the algorithms in [AC, CS1, CS2], *solve* is bit-vector-based. Each state in $\mathcal{S}$ is assigned a vector containing one bit for each variable X_j in E. The components of the bit vectors represent the current approximation $\{X_1 \mapsto S_1, \ldots, X_n \mapsto S_n\} \in (2^S)^{\{X_1,\ldots,X_n\}}$ to $\llbracket E \rrbracket$ as follows: $s \in S_i$ if and only if $s.X[i]$ is set to true. When the algorithm terminates, these bit-vectors components represent $\llbracket E \rrbracket$.

Given an equational system E, the algorithm works by first determining the hierarchical components $C_1, \ldots, C_m$ of E. It then processes each C_i in turn, beginning with C_1 and ending with C_m; $\llbracket C_i \rrbracket e_{i-1}$ is computed and stored in the relevant bit-vector components, and these results are then used as necessary in the computation of the meaning of subsequent hierarchical components. The algorithm terminates after C_m is completed. Given that each $\llbracket C_i \rrbracket e_{i-1}$ is computed properly, correctness follows from Theorem 4.11.

At the heart of the algorithm lies a method for computing $\llbracket C_i \rrbracket e_{i-1}$ for a hierarchical component C_i. This computation proceeds in two phases. During the first phase, the block list of C_i is computed, and bit-vector components are initialized appropriately. In the second phase, the block list is analyzed, beginning with the blocks in the lowest nesting level, and proceeding up to the highest nesting level. To process a block, the algorithm essentially invokes variants of procedures for computing greatest and least fixpoints used in 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 blocks for the lower nesting levels that are affected must be re-initialized and recomputed.

In this computation of $\llbracket C_i \rrbracket e_{i-1}$, one may identify two flows of information.

- **The flow of assumptions:**
  Our algorithm may be seen as "assumption-based": during the computation of the blocks 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 $E_{i,nd(C_i)}$ down to $E_{i,1}$.

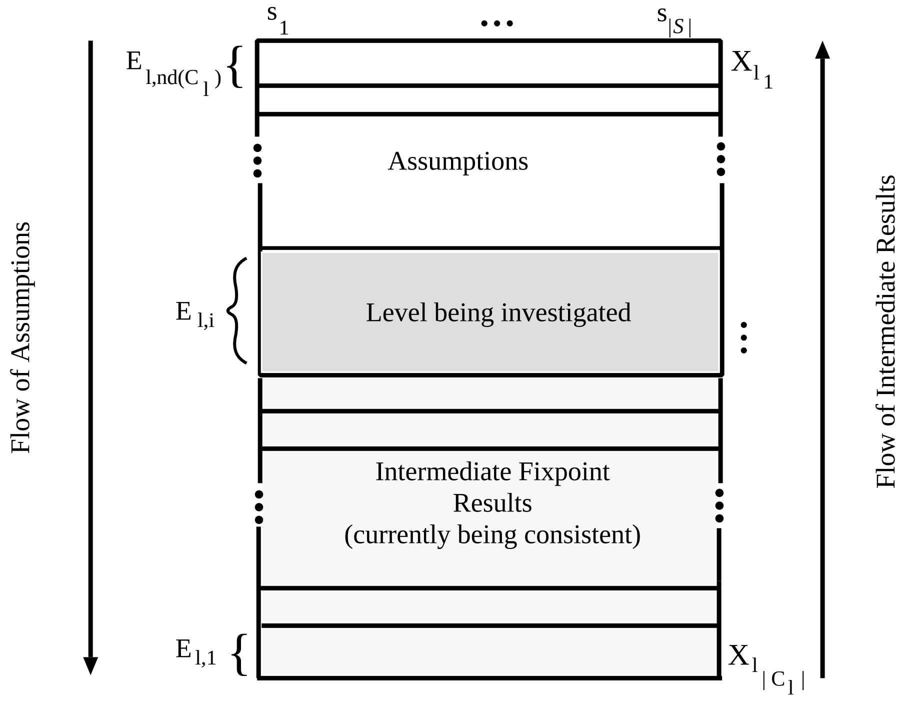

Figure 4: The Flows of Information in a Hierarchical Component C_l.

- **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_i with the bit-vectors corresponding to columns through the levels.

We now give a detailed description of *solve* and its main routines.

5.1 Routine *solve*

Input:

A labeled transition system $\mathcal{T}$ and a closed equational system E.

Output:

The labeled transition system $\mathcal{T}$ with states annotated with bit-vectors representing $[\![E]\!]$.

```
procedure solve ($\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle, E$)
    Build $G_E$;
    Compute hierarchical components $C_1, \ldots, C_m$;
    for $i := 1$ to $m$ do
        compSolve ($C_i$)
```

Figure 5: Routine *solve*.

Auxiliary Data Structures:

Each state in the labeled transition system has the following associated with it.

- An array $X[1..n]$ of bits. Bit $s.X[i]$ indicates whether state s belongs to the (current approximation to) $([\![E]\!])(X_i)$.
- An array $C[1..n]$ of counters. The counters support an efficient updating of bit-vector values for variables whose right-hand sides have a certain form.
- A bit $s.A$ for every atomic proposition A appearing in E. The bit $s.A$ indicates whether or not s satisfies A.

Pseudo-code for solve appears in Figure 5. The key step involves the call to routine *compSolve*, which computes the semantics of an individual hierarchical component. We now turn to a description of this procedure.

5.2 Routine *compSolve*

Routine *compSolve* determines the semantics of an individual hierarchical component of a given equational system. The procedure works in two phases; during the first, the block list for the component is computed and the relevant bit-vector components of each state are initialized appropriately, while the second consists of a scan through the block list, beginning with the lowest nesting level and proceeding upwards. The values computed for a block at one nesting level may necessitate the re-evaluation of lower nesting levels, owing to the fact that lower-level values may depend on assumptions made about the former values in the level; lower-level changes may in turn require the re-examination of values in the existing level. In order to determine efficiently which bit-vector values should change as a result of another change, the procedure maintains *work lists* for each block.

Given a block B_b, work list wl_b contains pairs of the form $\langle s, X_i \rangle$, where s is a state and X_i is a left-hand side in B_b. Intuitively, if $\langle s, X_i \rangle$ is on wl_b then the $X[i]$-component of s's bit vector has changed but has not yet been processed.

Figure 6 contains pseudo-code for *compSolve*. The procedure uses three subroutines: *initLevels*, which initializes the data structures for all levels below, and including, the level given to it as an argument; *propagate*, which updates the bit-vector components and the work list associated with the given block on the basis of values computed in lower blocks; and *processBlock*, which computes the semantics of the given block under the "assumptions" contained in the bit vectors for the higher nesting levels and the current

```
— Global values: $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle$, $E = \langle X_1 \lhd\!\rhd_1 \Phi_1, \ldots, X_n \lhd\!\rhd_n \Phi_n \rangle$.

procedure compSolve (C)
    Compute nesting levels $\mathcal{E} = \langle E_1, \ldots, E_m \rangle$;
    For each $E_l$, compute block list $\mathcal{B}_l = \langle B_{l,1}, \ldots, B_{l,b_l} \rangle$;
    Form $\mathcal{B} = \mathcal{B}_1 :: \cdots :: \mathcal{B}_m \equiv \langle B_1, \ldots, B_p \rangle$;
    for $b := 1$ to $p$ do $lev(b) :=$ nesting level to which $B_b$ belongs;
    initLevels ($m$);
    $b := 1$;
    while $b \leq p$ do
            propagate($b$);
            check := processBlock($b$);
            if check = true $\wedge \exists B_{b'}.lev(b') < lev(b) \wedge B_b \dashrightarrow B_{b'}$ then do
               initLevels ($lev(b) - 1$);
               $b := 1$
            else $b := b + 1$
```

Figure 6: Routine *compSolve*.

fixpoint approximation in lower blocks. This last procedure also returns a boolean indicating whether any changes have occurred in the block as a result of the processing. The next subsections give the details for these routines.

5.3 Routines *initLevels* and *propagate*

Routine *initLevels* has one parameter, l, which is the index of a nesting level. This procedure is responsible for initializing the bit-vector components and counters associated with the left-hand sides in levels $E_1, \ldots, E_l$. The pseudo-code for *initLevels* appears in Figure 7.

Procedure *initLevels* initializes levels from the "top down;" beginning with level l, it assigns values to bit-vector components and counters in the blocks in order of decreasing level. To initialize a block B_b, the routine first calls *initBlockPlain* that initializes max-components with *true* and min-components with *false*. Then it calculates the set *Avars*, which is the set of variables that are to be treated as "assumptions" for the level being processed. Note that *Avars* consists of all left-hand-side variables at higher levels within the current hierarchical component. Finally the routine *initBlock* is called; this procedure determines whether the block contains max- or min-equations and then issues the appropriate calls to the generic fixpoint routine *initGenVar*. The treatements of minimal blocks and maximal blocks are completely dual to one another, and the pseudo-code contained in Figure 7 exploits this by implementing them as calls to *initGenVar* with appropriate parameters to indicate what should be used as the bottom ($\bot$) and top ($\top$) values during the computation and what should be the relevant meet ($\sqcap$), join ($\sqcup$), universal modal ($\forall$), and existential modal ($\exists$) operations. In what follows we describe the operation of *initBlockPlain* and *initGenVar* on a maximal block. Given state s, variable index k, level index l, block index b, and assumption variable set *Avars*, this routine sets $s.X[k]$ and

```
— Global values: $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle, \mathcal{B} = \langle B_1, \ldots, B_p \rangle$, and $lev(b)$ and $wl_b$,
— where $1 \leq b \leq p$.

procedure initGenVar ($\bot, \top, \sqcap, \sqcup, \forall, \exists, s, k, b, Avars$);
    $tmp := s.X[k]$;
    case $\Phi_k$ is
        $A$                $\Rightarrow$ if $s.A = \bot$ then $s.X[k] := \bot$;
        $\neg A$           $\Rightarrow$ if $s.A = \top$ then $s.X[k] := \bot$;
        $X_i$              $\Rightarrow$ if $X_i \in Avars$ and $s.X[i] = \bot$ then $s.X[k] := \bot$;
        $X_i \sqcap X_j$   $\Rightarrow$ if $X_i \in Avars$ and $s.X[i] = \bot$ then $s.X[k] := \bot$;
                              else if $X_j \in Avars$ and $s.X[j] = \bot$ then $s.X[k] := \bot$;
        $X_i \sqcup X_j$   $\Rightarrow$ $s.C[k] := 2 - |\{ X_l \in \{X_k, X_j\} \mid X_l \in Avars \wedge s.X[l] = \bot \}|$;
                              if $s.C[k] = 0$ then $s.X[k] := \bot$;
        $\forall_a X_i$    $\Rightarrow$ if $X_i \in Avars$ and $\exists s'.\ s \xrightarrow{a} s' \wedge s'.X[i] = \bot$ then $s.X[k] := \bot$;
        $\exists_a X_i$    $\Rightarrow$ $s.C[k] := |\{ s' \mid s \xrightarrow{a} s' \}|$;
                              if $X_i \in Avars$ then $s.C[k] := s.C[k] - |\{ s' \mid s \xrightarrow{a} s' \wedge s'.X[i] = \bot \}|$;
                              if $s.C[k] = 0$ then $s.X[k] := \bot$;
    if $s.X[k] \neq tmp$ then add $\langle s, X_k \rangle$ to $wl_b$;

procedure initBlockPlain ($b$);
    $L := \{ X_k \mid X_k$ is a left-hand side variable in $B_b \}$;
    if $B_b$ is a max block then
      for each $s \in S, X_k \in L$ do $s.X[k] :=$ true;
    else
      for each $s \in S, X_k \in L$ do $s.X[k] :=$ false;

procedure initBlock ($b, Avars$);
    $L := \{ X_k \mid X_k$ is a left-hand side variable in $B_b \}$;
    if $B_b$ is a max block then
      for each $s \in S, X_k \in L$ do initGenVar (false, true, $\wedge, \vee, [], \langle\rangle, s, k, b, Avars$);
    else
      for each $s \in S, X_k \in L$ do initGenVar (true, false, $\vee, \wedge, \langle\rangle, [], s, k, b, Avars$);

procedure propagate ($b$);
    $Avars := \{ X_k \mid X_k$ is a left-hand side in some other block than $B_b \}$;
    initBlock ($b, Avars$);

procedure initLevels ($l$);
    $b := max\{ b \mid lev(b) = l \}$;
    for $b' := b$ downto 1 do
        initBlockPlain ($b$);
        $Avars := \{ X_k \mid X_k$ is a left-hand side in some $B_{b''}$ belonging to the same
                         component as $B_{b'}$ and $lev(b'') > lev(b') \}$;
        initBlock ($b, Avars$);
```

Figure 7: Routines *initLevels* and *propagate*.

$s.C[k]$ as follows.

- If the right-hand side of Φ_k is A or X_i, where X_i in an assumption variable, then $s.X[k]$ is assigned the same value as $s.A$ or $s.X[i]$. In case that Φ_k is of the form $\neg A$, $s.X[k]$ is assigned the dual value of $s.A$. If Φ_k is of the form X_i where X_i is not an assumption variable, then $s.X[i]$ is set to *true*.
- If Φ_k is of the form $X_i \wedge X_j$ or $[a]X_i$, then $s.X[i]$ is set to *true* unless:
  - in the case of $X_i \wedge X_j$ one of the conjuncts is an assumption variable and the corresponding bit-vector component for s is *false*; or
  - in the case of $[a]X_i$ with X_i being an assumption variable, and one of the a-derivatives of s has its X_i bit set to *false*.
- If Φ_k is of the form $X_i \vee X_j$ or $\langle a\rangle X_i$, then the counter $s.C[k]$ must also be initialized. In the first case, $s.C[k]$ is set to the number of its disjuncts that either are not assumption variables, or are assumption variables whose bits are *true*. In the second case, if X_i is not an assumption variable then $s.C[k]$ is set to the number of a-derivatives that s has; otherwise, it is set to the number of a-derivatives of s whose X_i bit is *true*. In either case, if $s.C[k]$ is zero then $s.X[k]$ is set to *false*.

Finally, if $s.X[k]$ has been initialized to *false* then the pair $\langle s, X_k\rangle$ is added to wl_b.

Procedure *propagate* is very similar to the block initialization of *initLevels*; the only differences are that no plain initialization is performed in order to maintain already computed values and that the set of "assumption variables" should be taken to be the set of all variables in other blocks so that values computed in lower-level blocks are taken into account.

5.4 Block Processing

The block-processing routine *processBlock* is responsible for computing fixpoints of individual blocks of equations. It first determines whether the block is a max block or a min block; then, it issues an appropriate call to a generic procedure, *processGenBlock*, which may be used to compute both maximum and minimum fixpoints given appropriate parameters indicating the bottom, top, meet, join and universal and existential modal operators. To process a max block, these should be the booleans false and true, and the logical operations $\wedge$, $\vee$, $[]$ and $\langle\rangle$, respectively, while for a min block the relevant parameters are true and false, and $\vee$, $\wedge$, $\langle\rangle$ and $[]$.

The fixpoint computation works by removing elements from the work list wl_b corresponding to the block B_b, propagating the effects of the changed component value to other bit-vector components associated to variables in B_b, and adding relevant pairs to wl_b.

In what follows we describe in more detail the processing that is performed on a max block; the handling of a min block is completely dual. When *processGenBlock* deletes a pair $\langle s, X_k\rangle$ from wl_b, it performs the following. (This account is essentially taken from [CS2].)

- For every X_j belonging to B_b such that $X_k \xrightarrow{\epsilon} X_j$ or $X_k \xrightarrow{\wedge} X_j$ if $s.X[j]$ is *true* the component $s.X[j]$ is set to *false* and $\langle s, X_j\rangle$ is added to wl_b for later processing.

```
— Global values: $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow \rangle, \mathcal{B} = \langle B_1, \ldots, B_p \rangle$, and $wl_b$, where $1 \le b \le p$.

procedure processGenBlock($\bot, \top, \sqcap, \sqcup, \forall, \exists, b$);
    $L := \{ X_j \mid X_j$ is a left-hand side of $B_b \}$;
    if $wl_b$ is not empty then $changed := true$;
    while $wl_b$ is not empty do
            choose $\langle s, X_k \rangle$ from $wl_b$;
            $wl_b := wl_b - \{\langle s, X_k \rangle\}$;
            for all $X_j \in L$ such that $X_k \xrightarrow{\epsilon} X_j \vee X_k \xrightarrow{\sqcap} X_j$ do
                if $s.X[j] = \top$ then
                    $s.X[j] := \bot$;
                    add $\langle s, X_j \rangle$ to $wl_b$;
            for all $X_j \in L$ such that $X_k \xrightarrow{\sqcup} X_j$ do
                $s.C[j] := s.C[j] - 1$;
                if $s.C[j] = 0$ then
                    $s.X[j] := \bot$;
                    add $\langle s, X_j \rangle$ to $wl_b$;
            for all $X_j \in L$ such that $X_k \xrightarrow{\forall_a} X_j$ do
                for all $s' \in \{ s' \mid s' \xrightarrow{a} s \}$ do
                    if $s'.X[j] = \top$ then
                        $s'.X[j] := \bot$;
                        add $\langle s', X_j \rangle$ to $wl_b$;
            for all $X_j \in L$ such that $X_k \xrightarrow{\exists_a} X_j$ do
                for all $s' \in \{ s' \mid s' \xrightarrow{a} s \}$ do
                    $s'.C[j] := s'.C[j] - 1$;
                    if $s'.C[j] = 0$ then
                        $s'.X[j] := \bot$;
                        add $\langle s', X_j \rangle$ to $wl_b$;
    return changed;

procedure processBlock ($b$);
    if $B_b$ is a max block then
        return processGenBlock (false, true, $\wedge, \vee$, [], $\langle\rangle$, $b$)
    else
        return processGenBlock (true, false, $\vee, \wedge$, $\langle\rangle$, [], $b$)
```

Figure 8: Block-processing Routines.

- For every X_j belonging to B_b such that $X_k \xrightarrow{\vee} X_j$, the counter $s.C[j]$ is decremented by one. If $s.C[j]$ is now 0, then none of the disjuncts on the right-hand side of X_j are satisfied by s, and s must be removed from the set associated with X_j. Accordingly, $s.X[j]$ is set to *false* and $\langle s, X_j \rangle$ is added to wl_b.

- For every X_j belonging to B_b with $X_k \xrightarrow{[a]} X_j$, each state s' having s as an a-derivative has its $X[j]$-component examined, and if it is *true* then it is changed to *false* and $\langle s', X_j\rangle$ is added to wl_b.

- For every X_j belonging to B_b with $X_k \overset{\langle a\rangle}{\rightarrow} X_j$, each counter $C[j]$ for each s' that has s as an a-derivative is decremented by one, and if it becomes 0 (meaning that s' now has no a-derivatives satisfying X_k), then $s'.X[j]$ is set to *false* and $\langle s', X_j\rangle$ is added to wl_b.

When the work list wl_b is empty for a given *max* block B_b, the bit-vector entries in each state that correspond to the variables in B_b contain the relevant fixpoint values. They are guaranteed not to change further during the fixpoint computation of the currently investigated level because of the hierarchical dependencies between the blocks and the order of block processing from the lowest to the highest.

6 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 for formulas while the environment is represented in the bit vectors. Together with Theorem 4.11 this enables us to prove the following theorem.

Theorem 6.1 (Correctness)
Let $\mathcal{T} = \langle\mathcal{S}, Act, \rightarrow\rangle$ be a finite labeled transition system and $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n\rangle$ be a closed, simple equational system. Then, the model-checking algorithm terminates with a bit-vector annotation that represents $[\![E]\!]$.

Proof.
On the basis of Theorem 4.11, *solve* is correct if *compSolve* can be shown to compute correctly the semantics of a hierarchical component C. Let $B_1, \ldots, B_k$ be the block list of C. Let C_{k-1} represent C with equations in B_k removed. On the basis of Theorem 4.18, it suffices to show that *compSolve* computes either $\nu G^C_{\nu,k}$ or $\mu G^C_{\mu,k}$, depending on whether B_k is a max block or a min block. On the basis of Theorems 2.3 and 4.8, it follows that *processGenBlock* computes correct extremal fixpoints for B_k, and hence $G^C_{\nu(\mu),k}$, while the resetting and recalculation of lower-level blocks after the computation of successive fixpoints for B_k leads to the proper calculation of the relevant extremal fixpoint of this function. Thus the algorithm is correct. □

The following theorem states our complexity result, where $|\mathcal{T}| = |\mathcal{S}| + |\rightarrow|$, $|E|$ is the number of equations in E and $rad(E)$ is its reduced alternation depth. Note that for all equational systems E $rad(E) \leq ad(E)$.

Theorem 6.2 (Complexity)
Let $E = \langle X_1 \diamond_1 \Phi_1, \ldots, X_n \diamond_n \Phi_n\rangle$ be a simple, closed equational system with $rad(E) \geq 1$, and $\mathcal{T} = \langle\mathcal{S}, Act, \rightarrow\rangle$ be a finite-state transition system. Then the worst-case time complexity of the model-checking algorithm solve is

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

Proof. Let $C_1, \ldots, C_m$ be the hierarchically ordered collection of components of E, which can be generated in $O(|E|)$ time (cf. Theorem 4.11). It is possible to check in $O(|E|)$ time whether E is alternation free or not (cf. Theorem 4.16 (1)) and, in the case of $rad(E) > 1$, to generate the nesting levels and blocks in $O(|E|^2)$ time (cf. Theorem 4.16 (2)). Because of the hierarchical dependencies between the hierarchical components the complexity of our model-checking algorithm is bound by the complexity for computing the semantics of the hierarchical component with the highest nesting depth. The additional linear factor for computing the other hierarchical components can be neglected. Therefore, the worst case occurs when E just has the trivial hierarchical component $C_1 = E$ itself. Thus we may assume that $E = C_1$ and $nd(C_1) = rad(E)$, which allows us to omit the index of the hierarchical component currently being investigated.

Let $\langle B_1, \ldots, B_p \rangle$ be the block list of E in hierarchical order of computation. Then, the worst case time complexity of the algorithm can be estimated by

$$\sum_{b=1}^{p} C(b) * R(b)$$

where $C(b)$ are the costs to reach a fixpoint over the bit-vectors of block B_b that is consistent with the assumptions of higher-level blocks and with all lower blocks. $R(b)$ represents the number of times such computations may be required in the worst case depending on the changes in higher levels. $C(b)$ is composed as follows.

$$C(b) = C_{init}(b) \; + \; C_{prop}(b) \; + \; C_{fix}(b)$$

where $C_{init}(b)$ denotes the costs for initializing the bit-vector values and worklists of block B_b taking the assumptions of higher levels into account, $C_{prop}(b)$ are the costs for propagating the results of lower blocks into block B_b, and $C_{fix}(b)$ represents the costs for computing a fixpoint over B_b that is consistent with the assumptions in higher-level blocks and with the fixpoints over the lower blocks.

Our estimation relies on the following numerical results for $0 \leq k \leq rad(E)$.

(1) $|\mathcal{S}| \geq 3$ implies

(2) $\sum_{i=1}^{k} |E_i| \;\leq\; |E|$ implies $\prod_{i=1}^{k} |E_i| \;\leq\; \left(\frac{|E|}{k}\right)^k$

(3) $\left(1 + \frac{1}{n}\right)^n$ asymmptotically approximates e.

(1) can be proved by straightforward induction, (2) is the result of determining the minimum of a k-dimensional function, and (3) is a well-known fact from mathematical analysis.

In what follows, we will use c_i to denote the constant factors that arise during the estimation. Note that for propagation and fixpoint computation we have to distinguish the cases of block B_b belonging to the lowest nesting level and B_b belonging to a higher nesting level. Thus, let $B_1, \ldots, B_i$ belong to the lowest nesting level and $B_{i+1}, \ldots, B_p$ belong to higher nesting levels. We then have:

$C_{init}(b)$: The initialization costs for assigning a value to each bit-vector component corresponding to variables in B_b and updating the worklist by taking the assumptions of higher levels into account (performed by the appropriate instance of *initGenVar* called by *initBlock*) can be estimated by

$$C_{init}(b) \leq c_1 * |\mathcal{T}| * |B_b|$$

as each state/variable pair has to be investigated at most once regarding all its transitions.

$C_{prop}(b)$: Distinguishing the cases of B_b belonging to the lowest or to a higher nesting level we get the following.

1. Let $1 \leq b \leq i$ such that B_b belongs to the lowest nesting level. The costs for propagating the values from all lower blocks can be estimated by

   $$C_{prop}(b) \leq c_2 * |\mathcal{T}| * |B_b|$$

   as each state/variable pair has to be investigated at most once regarding all its transitions. As there is no resetting and recomputation of lower blocks this propagation is performed exactly once to update the worklists according to the approximations in the lower blocks.
2. Let $i < b \leq p$ such that B_b does not belong to the lowest nesting level. As a change in block B_b necessiates the resetting and recomputation of lower blocks propagation has to be performed as often as changes may occur in B_b. Thus, the costs may be estimated by

   $$\begin{aligned} C_{prop}(b) &\leq c_3 * |\mathcal{T}| * |B_b| * |B_b| * |\mathcal{S}| \\ &= c_3 * |\mathcal{T}| * |B_b|^2 * |\mathcal{S}| \end{aligned}$$

$C_{fix}(b)$: The costs for computing a fixpoint for block B_b that is consistent with the assumptions in higher-level blocks and the current approximation in lower blocks can be estimated by $c_4 * |\mathcal{T}| * |B_b|$ as the monotinicity of fixpoint computation ensures that each state/variable pair is added at most once to the worklist.

1. In case of $1 \leq b \leq i$ such that B_b belongs to the lowest nesting such a fixpoint computation occurs exactly once and we have

   $$C_{fix}(b) \leq c_4 * |\mathcal{T}| * |B_b|$$

2. In case of $i < b \leq p$ such that B_b belongs to a higher nesting level fixpoint computation can be performed $|B_b| * |\mathcal{S}|$ in the worst case as that many changes may occur in the bit-vector associated with B_b. Each change requires resetting and recomputation of lower-level blocks that in turn require a new fixpoint computation for B_b such that we have

   $$\begin{aligned} C_{fix}(b) &\leq c_4 * |\mathcal{T}| * |B_b| * |B_b| * |\mathcal{S}| \\ &= c_4 * |\mathcal{T}| * |B_b|^2 * |\mathcal{S}| \end{aligned}$$

$C(b)$: Collecting the results of the last three estimations we have

$$\begin{aligned} C(b) &\leq c_5 * |\mathcal{T}| * |B_b| && \text{for } 1 \leq b \leq i \\ C(b) &\leq c_6 * |\mathcal{T}| * |B_b|^2 * |\mathcal{S}| && \text{for } i < b \leq p \end{aligned}$$

$R(b)$: For block B_b belonging to nesting level E_l consistency with all other blocks has to be established once plus the number of times bit vectors corresponding to variables in higher-level blocks change. For blocks in level $E_{rad(E)-1}$ there are at most $|E_{rad(E)}| * |\mathcal{S}|$ possible changes in higher levels and for blocks in $E_{rad(E)-2}$ at most $|E_{rad(E)-1}| * |\mathcal{S}| * |E_{rad(E)}| * |\mathcal{S}|$ possible changes in higher levels. A simple inductive extension yields

$$\begin{aligned} R(b) &\leq \quad 1 + |\mathcal{S}|^{rad(E)-l} * \prod_{j=l+1}^{rad(E)} |E_j| \\ &\leq c_7 * |\mathcal{S}|^{rad(E)-l} * \prod_{j=l+1}^{rad(E)} |E_j| \end{aligned}$$

for B_b belonging to level l.

Collecting the results we obtain:

$$\begin{aligned}
& \sum_{b=1}^{p} C(b) * R(b) \\
= \; & \sum_{b=1}^{i} C(b) * R(b) + \sum_{b=i+1}^{p} C(b) * R(b) \\
\leq \; & c_8 * |\mathcal{T}| * |E_1| * |\mathcal{S}|^{rad(E)-1} * \prod_{j=1+1}^{rad(E)} |E_j| \; + \; \sum_{l=2}^{rad(E)} c_9 * |\mathcal{T}| * |E_l|^2 * \; |\mathcal{S}|^{rad(E)-l+1} * \prod_{j=l+1}^{rad(E)} |E_j| \\
\leq \; & c_8 * |\mathcal{T}| * |\mathcal{S}|^{rad(E)-1} * \prod_{j=1}^{rad(E)} |E_j| \; + \; \sum_{l=2}^{rad(E)} c_9 * |\mathcal{T}| * |E_l|^2 * |\mathcal{S}|^{rad(E)-l+1} * \prod_{j=l+1}^{rad(E)} |E_j| \\
\leq \; & c_8 * |\mathcal{T}| * |\mathcal{S}|^{rad(E)-1} * \prod_{j=1}^{rad(E)} |E_j| \; + \; c_9 * |\mathcal{T}| * |E| * \sum_{l=2}^{rad(E)} |\mathcal{S}|^{rad(E)-l+1} * \prod_{j=l}^{rad(E)} |E_j| \\
(2) \quad \leq \; & c_{10} * |\mathcal{T}| * |\mathcal{S}|^{rad(E)-1} * \left(\frac{|E|}{rad(E)} \right)^{rad(E)} \; + \\
& c_{11} * |\mathcal{T}| * |E| * \sum_{l=2}^{rad(E)} |\mathcal{S}|^{rad(E)-l+1} * \left(\frac{|E|}{rad(E)-l+1} \right)^{rad(E)-l+1} \\
\leq \; & c_{10} * |\mathcal{T}| * |E| * \left(|\mathcal{S}| * \frac{|E|}{rad(E)} \right)^{rad(E)-1} + c_{11} * |\mathcal{T}| * |E| * \sum_{l=2}^{rad(E)} |\mathcal{S}|^{l-1} * \left(\frac{|E|}{l-1} \right)^{l-1} \\
= \; & c_{10} * |\mathcal{T}| * |E| * \left(|\mathcal{S}| * \frac{|E|}{rad(E)} \right)^{rad(E)-1} + c_{11} * |\mathcal{T}| * |E| * \sum_{l=1}^{rad(E)-1} |\mathcal{S}|^{l} * \left(\frac{|E|}{l} \right)^{l} \\
(1) \quad \leq \; & c_{10} * |\mathcal{T}| * |E| * \left(|\mathcal{S}| * \frac{|E|}{rad(E)} \right)^{rad(E)-1} + c_{11} * 2 * |\mathcal{T}| * |E| * \left(|\mathcal{S}| * \frac{|E|}{rad(E)-1} \right)^{rad(E)-1} \\
\leq \; & c_{12} * |\mathcal{T}| * |E| * \left(|\mathcal{S}| * \frac{|E|}{rad(E)-1} \right)^{rad(E)-1}
\end{aligned}$$

Thus it remains to show that

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

This is done by estimating the quotient of the arguments, which easily reduces to

$$\left(\frac{rad(E)}{rad(E)-1}\right)^{rad(E)-1}$$

in the following way.

$$\begin{aligned}
\left(\frac{rad(E)}{rad(E)-1}\right)^{rad(E)-1} &= \left(\frac{rad(E)-1}{rad(E)-1}+\frac{1}{rad(E)-1}\right)^{rad(E)-1} \\
&= \left(1+\frac{1}{rad(E)-1}\right)^{rad(E)-1} \\
(3) &\leq e
\end{aligned}$$

This observation completes the proof of our complexity result. □

7 An Example

In this section we illustrate our algorithm with an example taken from [SW]. Consider the closed 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 closed equational system has the following form.

$$E \quad = \left[\begin{array}{lll} X_1 & \rightarrow & X_2 \\ X_2 & \leftarrow & [a]X_3 \\ X_3 & \leftarrow & X_4 \vee X_2 \\ X_4 & \leftarrow & X_5 \wedge X_1 \\ X_5 & \leftarrow & A \end{array}\right]$$

The transition system $\mathcal{T}$ we want to investigate is the triple $\langle \mathcal{S}, Act, \rightarrow \rangle$, where $\mathcal{S} = \{s, t, u, v\}$, $Act = \{a\}$ 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 $v \xrightarrow{a} v$. A graphical representation of $\mathcal{T}$ is given in Figure 10.

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 $s.X[1..5]$, $t.X[1..5]$, $u.X[1..5]$ and $v.X[1..5]$ we need some auxiliary data structures for investigating the levels: the counters $s.C[1..5]$, $t.C[1..5]$, $u.C[1..5]$ and $v.C[1..5]$, where $x.C[i]$ maintains a count of the number of components $y.X[j]$ that may change until $x.X[i]$ has to be flipped, and furthermore we need the fields recording whether the states satisfy the atomic formula A; so $s.A = ff$, while $t.A = u.A = v.A = tt$. The main routine *solve* then builds the dependency graph shown in Figure 9 and determines the hierarchical components C_1 holding the last equation and C_2 containing the remaining equations. Then it calls *compSolve*(C_1) which calls *initLevels*(1) to initialize the only block holding the last equation according to the valuation. Propagation and processing do not change anything. *compSolve*(C_2) first computes the nesting levels E_1 holding the second, 37

Figure 9: The Dependency Graph.

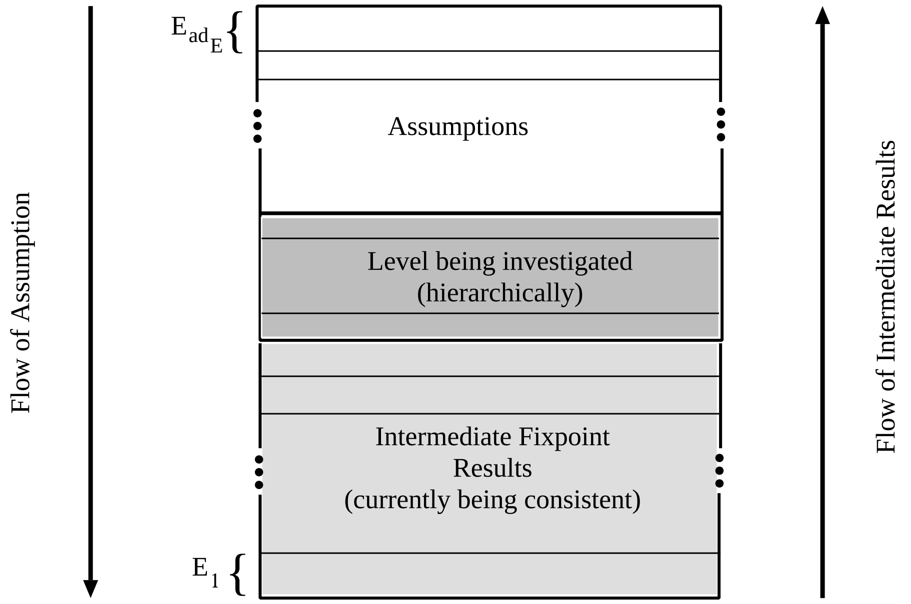

Figure 10: The Example Transition System.

third and fourth equation and E_2 holding the first equation. The block list $\langle B_1, B_2, B_3\rangle$ is derived such that B_1 contains the fourth equation, B_2 contains the second and third equation, and B_3 contains the first equation.

In what follows we highlight the changes made to the data structure due to the routines *initLevels*, *propagate* and *processBlock* step by step. Note in particular the change of intermediate results in B_1 and B_2 because of changing assumptions in B_3.

- After the computation of C_1 and the initialization of the levels in C_2 by calling *initLevels*(2) we have

	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*	*ff*	*ff*	*ff*
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	1	1	1	1
C_5	/	/	/	/

  with all worklists being empty. Then block B_1 is processed. Propagation of the fixpoint results in C_1 updates the worklists and counter as follows

	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*	$\boxed{tt}$	$\boxed{tt}$	$\boxed{tt}$
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	2	1
C_3	/	/	/	/
C_4	1	$\boxed{0}$	$\boxed{0}$	$\boxed{0}$
C_5	/	/	/	/

$wl_1 =$
$\{\langle t, X_4\rangle, \langle u, X_4\rangle, \langle v, X_4\rangle\}$

Processing B_1 just empties the worklist without changing the bit-vectors and counters. Processing B_2 starts with the propagation leading to

	s	t	u	v
X_1	*tt*	*tt*	*tt*	*tt*
X_2	*ff*	*ff*	*ff*	*ff*
X_3	*ff*	$\boxed{\mathit{tt}}$	$\boxed{\mathit{tt}}$	$\boxed{\mathit{tt}}$
X_4	*ff*	*tt*	*tt*	*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
C_5	/	/	/	/

$wl_2 =$
$\{\langle t, X_3\rangle, \langle u, X_3\rangle, \langle v, X_3\rangle\}$

Processing block B_2 now successively deletes $\langle t, X_3\rangle, \langle u, X_3\rangle$ and $\langle v, X_3\rangle$ from the worklist wl_2 and considers their influence. This provides the following intermediate results:

	s	t	u	v
X_1	*tt*	*tt*	*tt*	*tt*
X_2	*ff*	$\boxed{\mathit{tt}}$	*ff*	$\boxed{\mathit{tt}}$
X_3	*ff*	*tt*	*tt*	*tt*
X_4	*ff*	*tt*	*tt*	*tt*
X_5	*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
C_5	/	/	/	/

$wl_2 =$
$\{\langle t, X_2\rangle, \langle v, X_2\rangle\}$

The deletion of both pairs and the following processing of $X_2 \overset{\vee}{\to} X_3$ does not change the bit vectors as $t.X[3]$ and $v.X[3]$ are already *true*. Thereby, the processing of B_2 is completed and as no blocks in lower levels exist there is no resetting and recomputation because of the changes.

- Now B_3 is processed. First *propagate*(3) is called to adapt the bit-vector values to changes in the already processed blocks B_1 and B_2. This leads to a flip of $s.X[1]$ and $u.X[1]$ to *false* as $s.X[2] = u.X[2] = \mathit{ff}$.

	s	t	u	v
X_1	$\boxed{\mathit{ff}}$	*tt*	$\boxed{\mathit{ff}}$	*tt*
X_2	*ff*	*tt*	*ff*	*tt*
X_3	*ff*	*tt*	*tt*	*tt*
X_4	*ff*	*tt*	*tt*	*tt*
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	1	0	1	0
C_3	/	/	/	/
C_4	1	0	0	0
C_5	/	/	/	/

$wl_3 =$
$\{\langle s, X_1\rangle, \langle u, X_1\rangle\}$

Calling *processBlock*(3) just empties the worklist and does not cause any further changes in the data structures but the values for B_1 and B_2 have to be reinitialized and recomputed because of changes in B_3.

- *initLevels*(1) yields

	s	t	u	v
X_1	*ff*	*tt*	*ff*	*tt*
X_2	*ff*	$\boxed{\mathit{ff}}$	*ff*	$\boxed{\mathit{ff}}$
X_3	*ff*	$\boxed{\mathit{ff}}$	$\boxed{\mathit{ff}}$	$\boxed{\mathit{ff}}$
X_4	*ff*	$\boxed{\mathit{ff}}$	$\boxed{\mathit{ff}}$	$\boxed{\mathit{ff}}$
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	$\boxed{2}$	$\boxed{1}$	$\boxed{2}$	$\boxed{1}$
C_3	/	/	/	/
C_4	$\boxed{2}$	$\boxed{1}$	$\boxed{2}$	$\boxed{1}$
C_5	/	/	/	/

Propagating and processing of B_1 and B_2 results in

	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*	$\boxed{tt}$	*ff*	$\boxed{tt}$
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	$\boxed{1}$	1	$\boxed{1}$	$\boxed{0}$
C_3	/	/	/	/
C_4	2	$\boxed{0}$	$\boxed{1}$	$\boxed{0}$
C_5	/	/	/	/

- Propagating the effects of changes in B_1 and B_2 to B_3 by processing $X_2 \xrightarrow{\varepsilon} X_1$ yields a flip of $t.X[1]$ to *false* and the insertion of $\langle t, X_1\rangle$ to the worklist wl_3. Computing the fixpoint over B_3 by calling *processBlock*(3) empties the worklist. Due to changes in B_3 the lower two blocks have to be reinitialized and recomputed.
- Reinitailizing B_1 and B_2 and reprocessing them results in

	s	t	u	v
X_1	*ff*	*ff*	*ff*	*tt*
X_2	*ff*	*ff*	*ff*	*tt*
X_3	*ff*	*ff*	*ff*	*tt*
X_4	*ff*	*ff*	*ff*	*tt*
X_5	*ff*	*tt*	*tt*	*tt*

	s	t	u	v
C_1	/	/	/	/
C_2	2	1	1	0
C_3	/	/	/	/
C_4	2	1	1	0
C_5	/	/	/	/

Finally, B_3 is shown to be consistent with all lower blocks as the propagation does not change anything. Therefore, the algorithm terminates and we obtain $[\![\Phi]\!] = \{v\}$, as $v.X[1] = tt$ and the first bit-vector components of all other states are *false*. This reflects the intuition, because v is the only state for which A is infinitely often satisfied along all a-paths.

8 Conclusions and Future Work

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

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

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 $\frac{|E|}{rad(E)}$, which is important for formulas with high alternation depth.

- Instead of considering the notion of alternation depth as introduced by Emerson and Lei **??** we consider the improved notion of reduced alternation depth which is guaranteed to be smaller.

In [A] Andersen sketches an $O(|\mathcal{S}| * |T|^{ad(\Phi)-1} * |\Phi|^{ad(\Phi)})$ algorithm for the full μ-calculus, which improves on Emerson and Lei's result, $O((|\mathcal{T}| * |E|)^{ad(E)+1})$. Andersen's algorithm differs from ours in that it is tailored to the μ-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.

[CP] Cai, J., and R. Paige. “Program Derivation by Fixed Point Computation.” *Science of Computer Programming*, v. 11, 1988/89, pp. 197–261.

[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.

[CKS] Cleaveland, R., M.Klein and B. Steffen. “Faster Model Checking for the Modal Mu-Calculus.” In Proceedings CAV’92, 1992

[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.

```
procedure $proc(\Phi, \lhd\!\rhd, i)$;
    case $\Phi$ is
        $X$                          $\Rightarrow$ if $X$ is $X_j$ for some $j \leq i$ then return $(X, \langle\rangle, i)$
                                         else return $(X_i, \langle X_i \lhd\!\rhd X\rangle, i+1)$
        $A, \neg A, \neg X$          $\Rightarrow$ return $(X_i, \langle X_i \lhd\!\rhd \Phi\rangle, i+1)$;
        $\Phi_1 \; binop \; \Phi_2$  $\Rightarrow$ $(Y, E_1, i_1) := proc(\Phi_1, \lhd\!\rhd, i+1)$;
                                         $(Z, E_2, i_2) := proc(\Phi_2, \lhd\!\rhd, i_1)$;
                                         return $(X_i, \langle X_i \lhd\!\rhd Y \; binop \; Z\rangle :: E_1 :: E_2 \, , \, i_2)$;
        $modop \, \Phi_1$            $\Rightarrow$ $(Y, E_1, i_1) := proc(\Phi_1, \lhd\!\rhd, i+1)$;
                                         return $(X_i, \langle X_i \lhd\!\rhd \, modop \, Y\rangle :: E_1 \, , \, i_1)$;
        $\sigma X.\Phi_1$            $\Rightarrow$ if $\sigma = \mu$ then $\lhd\!\rhd := \leftarrow$ else $\lhd\!\rhd := \rightarrow$;
                                         $(Y, E_1, i_1) := proc(\Phi_1[X := X_i] \, , \lhd\!\rhd, i+1)$;
                                         return $(X_i, \langle X_i \lhd\!\rhd Y\rangle :: E_1, i_1)$;

procedure $trans\ (\Phi)$;
    Compute FNF $\Phi'$ that is equivalent to $\Phi$;
    $(X, E, j) := proc(\Phi', \rightarrow, 1)$;
    return $(E)$;
```

Figure 11: The Function *trans*.

A The Transformation Function *trans*

Function *trans* translates μ-calculus formulas into simple equational systems by introducing equations for subformulas. To do so, it employs an auxiliary recursive routine, *proc*, that takes the following arguments.

- A formula Φ.
- A variable $\lhd\!\rhd \in \{\rightarrow, \leftarrow\}$ to indicate whether max- or min-equations should be generated.
- An integer variable i that is the index of the next available left-hand-side variable X_i.

proc returns a triple consisting of: a variable whose meaning is to correspond to Φ; an equational system that contains the definitions needed to define the semantics of the variable; and the index i of the next available new variable X_i. Usually, the variable returned coincides with first left-hand-side variable in the equation list.

The definitions of *proc* and *trans* appear in Figure 11. In *proc*, $binop \in \{\vee, \wedge\}$ and $modop \in \{[a], \langle a\rangle\}$. We also use $\langle\rangle$ to represent an empty equational system, $\langle X_i \lhd\!\rhd \Phi_i\rangle$ for an equational system consisting of a single equation, and $E_1 :: E_2$ to represent the equational system formed by concatenating E_1 and E_2.

Briefly, *proc* works by performing a case analysis on Φ, the formula to be processed. If Φ is a (negated) atomic formula or negated variable, then X_i and one equation, $X_i \lhd\!\rhd \Phi$, are returned. If Φ is a variable X, then a new equation should be generated only if no existing equation has X as its left-hand side. When this is the case, $\langle X_i \diamond X\rangle$ and X_i are returned; otherwise, $\langle\rangle$ and X are returned. If Φ has the form Φ_1 *binop* Φ_2, then *proc* is called recursively on Φ_1 and Φ_2 to generate variables and equation systems for these subformulas. *proc* then generates a new equation whose left-hand side consists of X_i and right-hand side contains *binop* applied to the variables returned by the recursive calls. It then returns X_i together with the equation system comprising the new equation concatenated with the equation systems produced by the recursive calls. Similar processing is done when Φ is a modal formula. Finally, when Φ is a fixpoint formula $\sigma X.\Phi'$ with $\sigma \in \{\mu, \nu\}$, Φ' (with X renamed to X_i) is processed recursively, with the value of $\diamond$ for the recursive call determined by σ.

The routine *trans* essentially calls *proc* and returns the resulting equation list. Note that the $\diamond$ argument supplied to *proc* is $\rightarrow$; it could equally well have been $\leftarrow$. Furthermore, *trans* generates some superfluous equations of the form $X_i op_i X_j$ that are omitted in the equational systems in Exapmle 4.1.

To argue for the correctness of *trans*, we first note that the list it returns satisfies the requirements of an equation list: each right-hand side is simple, and no variable is used more than once as a left-hand side.

We now turn to proving Theorem 4.9, which states that *trans* is semantics-preserving. The next lemma establishes that the equational lists returned by *proc* enjoy certain properties with respect to the formula given as an argument.

Lemma A.1 *For any formula FNF Φ, environment e and $i \geq 1$ such that X_k is not free in Φ for any $k \geq i$, if proc$(\Phi, \diamond, i) = (X, E, j)$ then $[\![\Phi]\!]e = e[E](X)$.*

Proof. The proof proceeds by induction on the length of Φ. The induction hypothesis states that for any Φ' shorter than Φ, environment e and $i \geq 1$ such that X_k is not free in Φ' for any $k \geq i$, if *proc*$(\Phi', \diamond, i) = (X, E, j)$ then $[\![\Phi']\!]e = e[E](X)$. The proof now consists of a case analysis on the form of Φ.

- $\Phi \equiv Y$ for some $Y \in \mathit{Var}$. If $Y \equiv X_j$ for $j \leq i$ then $E = \langle\rangle$ and the result is trivial. If $Y \equiv X_j$ for $j > i$ then $E \equiv \langle X_i \diamond Y\rangle$; it then follows that $e[E] = e[\{X_i \mapsto e(Y)\}]$ from the definition of $e[E]$ and Lemma 3.4(1). As $X \equiv X_i$ in this case, we have that $e[E](X) = e(Y) = [\![\Phi]\!]e$.

- $\Phi \equiv A, \neg A$ or $\neg Y$, where $A \in \mathcal{A}$ and $Y \in \mathit{Var}$. The argument for this case is similar to the one given above and is omitted.

- $\Phi \equiv \Phi_1 \, \mathit{binop} \, \Phi_2$. Let (Y_1, E_1, i_1) be the result returned by *proc*$(\Phi_1, \diamond, i+1)$ and (Y_2, E_2, i_2) be the result of *proc*$(\Phi_2, \diamond, i_1)$. By assumption, X_j does not appear free in Φ for any $j \geq i$; as this must also hold for Φ_1 and Φ_2, it follows that the induction hypothesis guarantees that $[\![\Phi_1]\!]e = e[E_1](Y_1)$ and that $[\![\Phi_2]\!]e = e[E_2](Y_2)$. Moreover, as X_j is not free in Φ_1 or Φ_2 for $j \geq i$, it follows from the structure of *proc* that $\Delta(E_1 :: E_2) = \Delta(E_1) :: \Delta(E_2)$ and that $e[E_1](Y_1) = e[E_1 :: E_2](Y_1)$ and $e[E_2](Y_2) = e[E_1 :: E_2](Y_2)$. Thus, $[\![\Phi]\!]e = [\![\Phi_1 \, \mathit{binop} \, \Phi_2]\!]e = [\![Y_1 \, \mathit{binop} \, Y_2]\!]e[[\![E_1 :: E_2]\!]]$. Now consider the equational list $E \equiv \langle X_i \diamond Y_1 \, \mathit{binop} \, Y_2\rangle :: E_1 :: E_2$. From the structure of *proc* and our assumption it follows that X_i does not appear free E_1 or E_2; therefore, for any $Y \not\equiv X_i$, $e[E](Y) = e[E_1 :: E_2](Y)$. From the definition of $\Delta(E)$, we also have that $\Delta(E)(X_i) \equiv \sigma X_i.(Y_1 \, \mathit{binop} \, Y_2)[E_1 :: E_2]$, where $\sigma = \nu$ if $\diamond = \rightarrow$ and $\sigma = \mu$ otherwise. In addition, on the basis of Lemma 3.4(1) and Corollary 3.5, it follows that $\llbracket \Delta(E)(X_i) \rrbracket e = \llbracket (Y_1\, \mathit{binop}\, Y_2)[\Delta(E_1 :: E_2)] \rrbracket e = \llbracket Y_1\, \mathit{binop}\, Y_2 \rrbracket e[E_1 :: E_2]$. We thus have that $e[E](X_i) = \llbracket \Delta(E)(X_i) \rrbracket e = \llbracket \Phi \rrbracket e$.

- $\Phi \equiv \mathit{modop}\, \Phi'$. The argument in this case follows the same lines as the previous case and is omitted.

- $\Phi \equiv \sigma X.\Phi'$. Let (Y', E', i') be the result returned by $\mathit{proc}(\Phi'[X := X_i], \diamond', i+1)$, where $\diamond'$ is $\leftarrow$ if σ is μ and $\rightarrow$ otherwise. Since Φ is in FNF, X_i must appear free in $\Phi'[X := X_i]$. The induction hypothesis and Corollary 3.5 guarantee that $\llbracket \Phi'[X := X_i] \rrbracket e = e[E'](Y') = \llbracket Y'[\Delta(E')] \rrbracket e$ for any e, and from the definition of the semantics of $\sigma X.\Phi'$ it follows that $\llbracket \sigma X_i.(\Phi'[X := X_i]) \rrbracket e = \llbracket \sigma X_i.(Y'[\Delta(E')]) \rrbracket e$. Now consider the equational list $E \equiv \langle X_i \diamond' Y' \rangle :: E'$. From the definition of $\Delta(E)$ it follows that $\Delta(E)(X_i) = \sigma X_i.(Y'[\Delta(E')])$. Thus $e[E](X_i) = \llbracket \sigma X_i.(Y'[\Delta(E')]) \rrbracket e = \llbracket \sigma X_i.(\Phi'[X := X_i]) \rrbracket e$. Since X_i does not appear free in Φ' by assumption, Lemma 3.4(2) ensures that $e[E](X_i) = \llbracket \sigma X_i.(\Phi'[X := X_i]) \rrbracket e = \llbracket \Phi \rrbracket e$.

$\square$

Theorem 4.9 is now a corollary.

Corollary A.2 (Theorem 4.9) *Let Φ be a formula, and let $\{ X_i \mid i \geq 1 \}$ not appear free in Φ. Then $\llbracket \Phi \rrbracket e = (\llbracket \mathit{trans}(\Phi) \rrbracket)e)(X_1)$.*

Proof. From the structure of *proc*, it follows that if $\mathit{proc}(\Phi, \rightarrow, 1) = (X, E, i)$, then E is nonempty and $X \equiv X_1$. Thus, on the basis of Lemma A.1, it follows that $\llbracket \Phi \rrbracket e = e[E](X_1) = (\llbracket E \rrbracket e)(X_1)$. $\square$