Evidence-Based Model Checking*

Li Tan and Rance Cleaveland

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400 USA
{tanli, rance}@cs.sunysb.edu
Phone: +1 703 534 6458

Abstract. This paper shows that different "meta-model-checking" analyses can be conducted ef£ciently on a generic data structure we call a *support set*. Support sets may be viewed as abstract encodings of the "evidence" a model checker uses to justify the yes/no answers it computes. We indicate how model checkers may be modi£ed to compute supports sets without compromising their time or space complexity. We also show how support sets may be used for a variety of different analyses of model-checking results, including: the generation of diagnostic information for explaining negative model-checking results; and certifying the results of model checking (is the evidence internally consistent?).

Keywords: model checking; diagnostic information; mu-calculus; temporal logic

1 Introduction

Temporal-logic model checking [CE81,QS82,CES86] refers to an array of techniques for automatically determining whether or not a system satis£es a property expressed in some temporal logic. Traditionally, model checkers have been viewed as decision procedures that return yes/no answers re¤ecting the "correctness" of the system being analyzed. However, researchers have also realized that the information collected by model checkers in order to compute their answers can also be of great interest to the users of model checkers. *Diagnostic information* [CGMZ95,Sti95] explaining answers to users represents one use of such information; others include coverage analysis [CKV01], vacuity checking [BBDER97] (is (part of) a formula "trivially true", and hence probably erroneous?), and result certi£cation [Nam01] (does the evidence collected indeed support the conclusion returned, i.e. can the model checker be trusted?).

Existing "meta-model-checking" research is generally model-checker dependent: routines utilize algorithm-speci£c information computed during model-checking and hence are tightly bound to the infrastructure of checkers being used. In this paper we propose a generic framework for the analysis of model-checking results that uses a uniform encoding of the *evidence* collected by a model checker as it executes. In particular, we show how this evidence may be abstractly encoded in a special data structure, called a *support set*, that existing model checkers may be easily modi£ed to generate. We then illustrate how support sets can be used to support different analyses of model-checking results in a model-checker independent fashion. Using our results, builders of model-checking tools can factor out diagnostic-information generation, or

* Research supported by NSF grants CCR-9988489 and CCR-0098037 and Army Research Of£ce grants DAAD190110003 and DAAD190110019.

justi£cation generation, or coverage analysis, from their model checkers and into special support-set analyzers computing the answers in question. The result is uncluttered model-checking code and an extensible implementation in which different support-set-based "meta-model-checking" analyzers may be added without modifying the underlying model-checking engine.

The rest of paper is organized as follows. Section 2 contains mathematical preliminaries, while Section 3 de£nes support sets. The section following illustrates how model checkers may be altered to compute support sets ef£ciently. The next few sections show how support sets may be used in support of different "meta-model-checking" analyses. Section 7 concludes and discusses related work.

2 Preliminaries

This section de£nes the system models and temporal logics used in the rest of the paper. In the remainder of the report we £x a set $\mathcal{A}$ of *atomic propositions*.

2.1 Kripke Structures and CTL*

De£nition 1. *A Kripke structure is a tuple $\langle S, s_I, \rightarrow, V\rangle$, with S the set of states, $s_I \in S$ the start state, $\rightarrow \subseteq S \times S$ is transition relation and $V : \mathcal{A} \rightarrow 2^S$ the valuation.*

A Kripke structure encodes a system's operational behavior, with S being the possible set of system states and $\rightarrow$ the (atomic) state transitions. For atomic proposition $A \in \mathcal{A}$, $V(A)$ indicates in which states A is true. We usually write $s \rightarrow s'$ in lieu of $\langle s, s'\rangle \in \rightarrow$.

Given Kripke structure $\langle S, s_I, \rightarrow, V\rangle$, a *path* from $s \in S$ is a maximal sequence $\sigma = s_0 s_1 \ldots$ where $s = s_0$ and $s_i \rightarrow s_{i+1}$ for all $i < |\sigma|$. (Here $|\sigma| = n$ if $\sigma = s_0 s_1 \ldots s_n$ and ∞ if σ is in£nite.) We use $\sigma[i]$ for s_i and $\sigma^{(i)}$ for $s_i s_{i+1} \ldots$ if $i \leq |\sigma|$.

CTL*. Formulas in the logic CTL* are given via the following grammar, where $A \in \mathcal{A}$.

$$\begin{aligned}\phi &::= A \mid \neg A \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{A}\,\psi \mid \mathsf{E}\,\psi \\ \psi &::= \phi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathsf{X}\,\psi \mid \psi\,\mathsf{U}\,\psi \mid \psi\,\mathsf{R}\,\psi\end{aligned}$$

We refer to the formulas generated from ϕ as *state formulas* and those from ψ as *path formulas*. The CTL* formulas consist of the state formulas. We call A and E *path quanti£ers* and the X, U, R *path modalities*. The sublogic CTL consists of those CTL* formulas in which every path modality is immediately preceded by a path quanti£er.

CTL* formulas are interpreted with respect to Kripke structures $T = \langle S, s_I, \rightarrow, V\rangle$ where $\rightarrow$ is *total*: for every $s \in S$ there exists $s' \in S$ with $s \rightarrow s'$. Given such a T, the semantics of CTL* formulas is given via a relation $\models_T$ associating states s in T to state formulas and paths σ in T to path formulas and which is de£ned below.

1. $s \models_T A (\neg A)$ iff $s \in V(A)$ ($s \notin V(A)$).
2. $s \models_T \phi_1 \wedge \phi_2$ ($\phi_1 \vee \phi_2$) iff $s \models_T \phi_1$ and (or) $s \models_T \phi_2$.
3. $s \models_T \mathsf{A}\psi$ ($\mathsf{E}\psi$) iff for every (some) path σ from s, $\sigma \models_T \psi$.
4. $\sigma \models_T \phi$, where ϕ is a state formula, iff $\sigma[0] \models \phi$.
5. $\sigma \models_T \psi_1 \wedge \psi_2$ ($\psi_1 \vee \psi_2$) iff $\sigma \models_T \psi_1$ and (or) $\sigma \models_T \psi_2$.
6. $\sigma \models_T \mathsf{X}\psi$ iff $\sigma^{(1)} \models \phi$.
7. $\sigma \models_T \psi_1 \mathsf{U} \psi_2$ iff for some $i \geq 0$, $\sigma^{(i)} \models \psi_2$ and $\sigma^{(j)} \models \psi_1$ for all $j < i$.
8. $\sigma \models_T \psi_1 \mathsf{R} \psi_2$ iff for all $i \geq 0$ $\sigma^{(i)} \models \psi_2$ or $\sigma^{(j)} \models \psi_1$ some $j < i$.

The release modality R is the dual of the until operator U. Intuitively, $\psi_1 \mathsf{R} \psi_2$ holds of a path if ψ_2 is kept true until "released" from this obligation by the truth of ψ_1.

2.2 The Modal Mu-Calculus and Boolean Equation Systems

We de£ne the modal mu-calculus and boolean equation systems by £rst giving a general account of £xpoint equation systems over complete lattices [Mad97].

Lattices and Environments A *complete lattice* is a partially ordered set $\langle Q, \sqsubseteq \rangle$ with the following property: every subset $Q' \subseteq Q$ has a least upper bound $\bigsqcup Q'$ in Q. It can be shown that arbitrary upper bounds $\mathrm{d}\, Q$ also exist and that any complete lattice has a unique least element $\perp$ and maximum element $\top$. In addition, the Tarski-Knaster theorem guarantees the existence of unique least and greatest £xpoints for any monotonic function $f : Q \to Q$. Given monotonic f, we write $\mu f \in Q$ for the least "solution" to $f(x) = x$ and $\nu f \in Q$ for the greatest. These £xpoints are characterized as follows.

$$\mu f = \bigsqcup \{ q \in Q \mid f(q) \sqsubseteq q \} \qquad \nu f = \mathrm{d} \{ q \in Q \mid q \sqsubseteq f(q) \}$$

Let $\langle Q, \sqsubseteq \rangle$ be a complete lattice and $\mathcal{X}$ be a £nite set of *variables*. Then an *environment* over $\mathcal{X}$ is a function from $\mathcal{X}$ to Q. We use $Q^{\mathcal{X}}$ to represent the set of all environments over $\mathcal{X}$. Environments constitute a complete lattice under the pointwise extension of $\sqsubseteq$ to $Q^{\mathcal{X}}$: $\theta \sqsubseteq \theta'$ if and only if for all $X \in \mathcal{X}, \theta(X) \sqsubseteq \theta'(X)$.

If $\theta \in Q^{\mathcal{X}}$ and $\mathcal{X}' \subseteq \mathcal{X}$, then $\theta | \mathcal{X}' \in Q^{\mathcal{X}'}$ is de£ned by $(\theta | \mathcal{X}')(X) = \theta(X)$ for all $X \in \mathcal{X}'$. If $\theta' \in Q^{\mathcal{X}'}$ then $\theta[\theta']$ denotes the environment obtained by *updating* θ by θ':

$$\theta[\theta'] = \begin{cases} \theta'(X) & \text{if } X \in \mathcal{X}' \\ \theta(X) & \text{otherwise} \end{cases}$$

Fixpoint Equation Systems We now develop a general framework for systems of equations over a complete lattice. Throughout the remainder of this subsection we £x a complete lattice $\langle Q, \sqsubseteq \rangle$ and a £nite set $\mathcal{X}$ of variables.

Syntax An *equation block* B is a set of equations $\{X_1 = f_1, \ldots, X_l = f_l\}$, where the f_i are monotonic functions in $Q^{\mathcal{X}} \to Q$, $\{X_1, \ldots X_l\} \subseteq \mathcal{X}$, and the X_i are distinct. We use $\mathit{lhs}(B) = \{X_1, \ldots, X_l\}$ for the left-hand side variables of B, and $\mathit{rhs}(B, X_i) = f_i$ for the right-hand side for X_i in B, respectively. The f_i are often represented syntactically as expressions involving free occurrences of the variables from $\mathcal{X}$. In this case, we use $\mathit{vars}(f_i)$ to refer to the set of variables occurring freely in f_i, and we de£ne $\mathit{vars}(B) = \mathit{lhs}(B) \cup \bigcup_{i=1}^{l} \mathit{vars}(f_i)$ as the *variables* in equation block B. We refer to the variables in $\mathit{lhs}(\mathcal{B})$ as *bound* and the variables in $\mathit{vars}(\mathcal{B}) - \mathit{lhs}(\mathcal{B})$ as *free*.

A *parity block* E has form $\langle \sigma, B \rangle$, where $\sigma \in \{\mu, \nu\}$ is a *parity indicator* and B is an equation block. We lift the notions lhs, rhs, vars, free variable, and bound variable to parity blocks in the straightforward manner.

A *£xpoint equation system* is a nonempty sequence $\mathcal{E} = E_1 \ldots E_m$ of parity blocks whose left-hand sides are pairwise disjoint. If $\mathcal{E}'$ is an equation system and E is a parity block whose left-hand side variables are disjoint from those in $\mathcal{E}'$ then we write $E :: \mathcal{E}'$ for the equation system obtained by adding E to the front of $\mathcal{E}'$. We use $\mathcal{E}^{(k)} = E_k E_{k+1} \cdots$ to refer to the subsequence of $\mathcal{E}$ starting from k-th parity block. The operations lhs, rhs, vars, etc., are generalized in the straightforward manner. We call $\mathcal{E}$ *closed* if every $X \in \mathit{vars}(\mathcal{E})$ is bound, i.e. an element of $\mathit{lhs}(\mathcal{E})$. We also de£ne $h_{\mathcal{E}}(X) = k$ when $X \in \mathit{lhs}(E_k)$ and refer to $h_{\mathcal{E}}(X)$ as the *depth* of X in $\mathcal{E}$. We write $h(X)$ when $\mathcal{E}$ is clear from context. We say X_i is *shallower* or *higher* than X_j

if $h(X_i) < h(X_j)$, and *deeper* (or *lower*) if $h(X_i) > h(X_j)$. If X is a left-hand side variable in $\mathcal{E}$ we de£ne $\sigma_{\mathcal{E}}(X)$ to be the parity of the unique parity block E in $\mathcal{E}$ such that $X \in \mathit{lhs}(E)$. We omit reference to $\mathcal{E}$ and write $\sigma(X)$ when $\mathcal{E}$ is clear from context.

In a £xpoint equation system $\mathcal{E}$, we say that X_i syntactically depends on X_j, written as $X_i \lhd X_j$, if $X_j \in vars(rhs(X_i))$. We write $\overset{*}{\lhd}$ for the transitive and re¤exive closure of $\lhd$; so $X_i \overset{*}{\lhd} X_j$ if there is a path of syntactic dependencies from X_i to X_j.

Semantics Let $\theta \in Q^{\mathcal{X}}$ be an environment; then a single block $B = \{X_1 = f_1, \ldots, X_l = f_l\}$, where $\mathcal{X}' = \{X_1, \ldots, X_l\}$, may be seen as inducing a function $f_{B,\theta} : Q^{\mathcal{X}'} \to Q^{\mathcal{X}'}$ mapping environments over $\mathcal{X}'$ to environments over $\mathcal{X}'$ as follows.

$$f_{B,\theta}(\theta') = (X_1 \mapsto f_1(\theta[\theta'])) \cdots [X_l \mapsto f_l(\theta[\theta'])]$$

That is, $f_{B,\theta}(\theta')$ returns an environment over $\mathcal{X}'$ in which each X_i is mapped to the result of evaluating f_i on environment $\theta[\theta']$. Note that in $\theta[\theta']$ θ' "overwrites" the values for the X_i in θ. Consequently, θ may be seen as providing values only for variables that are not in $\mathcal{X}'$. It follows from the monotonicity of the f_j that for any θ, $f_{B,\theta}$ is a monotonic function over $\mathcal{Q}^{\mathcal{X}'}$, and consquently, least and greatest £xpoints, $\mu f_{B,\theta}$ and $\nu f_{B,\theta}$, which are environments over $\mathcal{X}'$, exist. Given $\theta \in Q^{\mathcal{X}}$, we then de£ne the semantics of a parity block in terms of these £xed points: $[\![\langle\sigma, B\rangle]\!]\theta = \sigma f_{B,\theta}$. So $[\![\langle\sigma, B\rangle]\!]$ maps environments over $\mathcal{X}$ to environments over $\mathit{lhs}(B)$.

Fixpoint equation systems are now interpreted as follows. For an environment $\theta \in Q^{\mathcal{X}}$ and equation system $\mathcal{E}$ with $\mathit{lhs}(\mathcal{E}) = \mathcal{X}_{\mathcal{E}} \subseteq \mathcal{X}$ we de£ne a function $f_{\mathcal{E},\theta}$ that maps environments in $Q^{\mathcal{X}_{\mathcal{E}}}$ to environments in $Q^{\mathcal{X}_{\mathcal{E}}}$. We then use an appropriate £xpoint of this function to arrive at the meaning of $\mathcal{E}$. We de£ne $f_{\mathcal{E},\theta}$ by induction on the structure of $\mathcal{E}$. When $\mathcal{E} = \langle\sigma, B\rangle$ (i.e. $\mathcal{E}$ contains one block), then we take $f_{\mathcal{E},\theta} = f_{B,\theta}$. In this case $f_{\mathcal{E},\theta}$ is clearly monotonic, and we de£ne $[\![\mathcal{E}]\!]\theta = [\![\langle\sigma, B\rangle]\!]\theta$. When $\mathcal{E} = \langle\sigma, B\rangle :: \mathcal{E}'$ contains two or more blocks, $f_{\mathcal{E},\theta}$ is de£ned as

$$f_{\mathcal{E},\theta}(\theta') = ([\![\mathcal{E}']\!](\theta[\theta']))[f_{B,\theta[[\![\mathcal{E}']\!](\theta[\theta'])]}(\theta'|\mathcal{X}_B)],$$

where $\mathcal{X}_B = \mathit{lhs}(B)$. This expression may be understood via its subexpressions.

$[\![\mathcal{E}']\!](\theta[\theta'])$ is the environment de£ned by $\mathcal{E}'$ in environment θ updated with bindings in θ'. This environment assigns a "£xpoint value" to every left-hand variable in $\mathcal{E}'$.
$f_{B,\theta[[\![\mathcal{E}']\!](\theta[\theta'])]}$ is the function on environments de£ned by block B and the environment obtained by updating θ with the bindings in $\mathcal{E}'$.
$\theta'|\mathcal{X}_B$ is the subenvironment of θ' obtained by restricting variables to those that appear as left-hand sides in B.

It is easy to show that $f_{\mathcal{E},\theta}(\theta')$ is monotonic over the lattice $Q^{\mathcal{X}_{\mathcal{E}}}$ and hence has unique least and greatest £xpoints. We then de£ne $[\![\mathcal{E}]\!]\theta$ as: $[\![\langle\sigma, B\rangle :: \mathcal{E}']\!]\theta = \sigma f_{\langle\sigma,B\rangle::\mathcal{E}',\theta}$.

If $\mathcal{E}$ is closed then for any θ, θ' we have that $[\![\mathcal{E}]\!]\theta = [\![\mathcal{E}]\!]\theta'$. In this case we often omit reference to θ and write $[\![\theta]\!]$ for this (unique) environment.

We conclude this general treatment of £xpoint equation systems with a the de£nition of *alternation depth*. Here we adopt the convention that $max\emptyset = 0$.

De£nition 2. *Let $\mathcal{E} = E_1 E_2 \cdots E_m$ be an equation system and X a left-hand side variable in $\mathcal{E}$. Then the* alternation depth*, $ad(X)$ of X is given as:*

$$ad(X) = 1 + max\{ad(X') \mid \sigma(X) \neq \sigma(X'), h(X') < h(X), X \overset{*}{\lhd} X' \overset{*}{\lhd} X\}$$

The Modal Mu-Calculus In this paper we de£ne modal mu-calculus formulas using £xpoint equation systems whose right-hand sides are formulas built as follows.

$$f = A \mid \neg A \mid \bigwedge \mathcal{X}' \mid \bigvee \mathcal{X}' \mid \langle\rangle X \mid [] X$$

Here $A \in \mathcal{A}$ and $\mathcal{X}' \subseteq \mathcal{X}$. The lattice $\langle Q, \sqsubseteq \rangle$ used to interpret variables is given by £xing a Kripke structure $T = \langle S, s_I, \rightarrow, V \rangle$ and taking $Q = 2^S$ and $\sqsubseteq = \subseteq$. We adopt the usual semantics of modal formulas given below; note that $\theta \in (2^S)^{\mathcal{X}}$ maps variables to sets of states in T. For any f, $[\![f]\!]_T$ is a monotonic function from $(2^S)^{\mathcal{X}}$ to 2^S.

$$\begin{array}{rlrl}
[\![A]\!]_T\theta &= V(A) & [\![\neg A]\!]_T\theta &= S - V(A) \\
[\![\bigvee \mathcal{X}']\!]_T\theta &= \bigcup\{\theta(X) \mid X \in \mathcal{X}'\} & [\![\bigwedge \mathcal{X}']\!]_T\theta &= \bigcap\{\theta(X) \mid X \in \mathcal{X}'\} \\
[\![\langle\rangle X]\!]_T\theta &= \{s \mid \exists s \rightarrow s'.\ s' \in \theta(X)\} & [\![[] X]\!]_T\theta &= \{s \mid \forall s \rightarrow s'.\ s' \in \theta(X)\}
\end{array}$$

Mu-calculus £xpoint equation systems in essence de£ne a collection of formulas, one for each $X \in lhs(\mathcal{E})$. This notation is clumsy for users, but more user-friendly logics such as CTL, LTL and CTL* may be translated ef£ciently into it [Dam94,BC96b].

Boolean Equation Systems Boolean equation systems are £xpoint equation systems de£ned over the boolean lattice $\langle \{\mathbf{0}, \mathbf{1}\}, \sqsubseteq \rangle$, where $\mathbf{0}$ and $\mathbf{1}$ are the boolean values "false" and "true", respectively, and $\mathbf{0} \sqsubset \mathbf{1}$. In this setting environments may be viewed as characteristic functions of subsets of $\mathcal{X}$, so we use set operators $\cup$, $\cap$, and $-$ on such environments. The right-hand sides of equations are the formulas given by the following, where $\mathcal{X}' \subseteq \mathcal{X}$.

$$f := \bigvee \mathcal{X}' \mid \bigwedge \mathcal{X}'$$

We often write tt for $\bigwedge \emptyset$ and ff for $\bigvee \emptyset$. The de£nition of $[\![f]\!]\theta$ is standard: $[\![\bigvee \mathcal{X}']\!]\theta = \mathbf{1}$ iff $\mathcal{X}' \cap \theta \neq \emptyset$, and $[\![\bigwedge \mathcal{X}']\!]\theta = \mathbf{1}$ iff $\mathcal{X}' \subseteq \theta$.

Boolean equation systems may be derived from mu-calculus equation systems and Kripke structures. Intuitively, this is done by assigning a boolean variable to each state / mu-calculus variable pair; the boolean variable is intended to indicate whether or not the state is in the set of states associated with the mu-calculus variable. The resulting boolean equation system has alternation depth no greater than the mu-calculus equation system from which it is derived.

3 Support Sets

When a model-checking problem is encoded as a boolean equation system, the goal is typically to determine the value of a single distinguished variable ("does the start state satisfy the formula?"). A support set stores the evidence for such a variable's value as an abstract "proof" recording how values of variables depend on values of other variables.

De£nition 3 (Support Set). *Let $\mathcal{E} = E_1 \dots E_m$ be a closed boolean equation system with $\mathcal{X} = lhs(\mathcal{B})$, let $X \in \mathcal{X}$, and let $r \in \{\mathbf{0}, \mathbf{1}\}$. Then a* support set *for r and X is a triple $\Gamma = \langle r, X, \Xi \rangle$, where $\Xi : \mathcal{X} \rightarrow 2^{\mathcal{X}}$ is a partial function such that $\Xi(X)$ is de£ned and such that the following properties hold for each X_i where $\Xi(X_i)$ is de£ned ($X_i \underset{\Gamma}{\rightarrow} X_j$ if $(\Xi(X_i))(X_j) = r$).*

1.(Direct Inference) $[\![f_i]\!](\Xi(X_i)) = r$

II.(Inclusion) If $X_i \xrightarrow[\Gamma]{} X_j$, then $\Xi(X_j)$ is de£ned.
III.(Circularity Restriction) If there exists a loop $\rho \equiv X_{p_1} \xrightarrow[\Gamma]{} \ldots \xrightarrow[\Gamma]{} X_{p_1}$ and X_i is the shallowest variable on ρ, then $(r = \mathbf{1} \Rightarrow \sigma_i = \nu) \wedge (r = \mathbf{0} \Rightarrow \sigma_i = \mu)$.

If $\Gamma = \langle r, X, \Xi \rangle$ then we call r the *support value* and X the *support variable* of Γ.

Support sets may be understood as follows. Recall that environments over the boolean lattice are isomorphic to sets of variables. Thus Ξ may be seen as associating an environment to each variable X_i on which it is de£ned. The existence of an edge $X_i \xrightarrow[\Gamma]{} X_j$ indicates a dependency of the value of X_i on X_j. Thus Condition I asserts that if $\Xi(X_i)$ is de£ned then under the interpretation $\Xi(X_i)$ of its variables, f_i evaluates to r, the boolean result of the support set. Condition II requires all variables on which X_i depends to be in the domain of Ξ. The last condition imposes restrictions on cyclic dependencies: the parity of the "shallowest" variable on the cycle must be consistent with r.

We may de£ne an environment $g(\Gamma)$ for $\Gamma = \langle r, X, \Xi \rangle$ as follows.

$$(g(\Gamma))(X_i) = \begin{cases} r & \text{if } \Xi(X_i) \text{ is de£ned} \\ \bar{r} & \text{otherwise} \end{cases}$$

Theorem 1 states that the environment de£ned by a support set constitues a *partial model* of $\mathcal{E}$ in the following sense: if $r = 1$ then $g(\Gamma) \subseteq [\![\mathcal{E}]\!]$, and if $r = 0$ then $g(\Gamma) \cap [\![\mathcal{E}]\!] = \emptyset$. Since $\Xi(X)$ is always de£ned, it follows that $[\![\mathcal{E}]\!](X) = r$.

Theorem 1. *Let $\mathcal{E}$ be a closed boolean equation system with $X \in lhs(\mathcal{E})$ and a support set $\Gamma = \langle r, X, \Xi \rangle$. Then $g(\Gamma)$ is a partial model for $\mathcal{E}$.*

The next theorem guarantees the existence of support sets.

Theorem 2. *Given closed boolean equation system $\mathcal{E}$ and $X \in lhs(\mathcal{E})$, let $r = [\mathcal{E}](X)$. Then there exists a support set for $\mathcal{E}$ with support value r and support variable X.*

Support Sets for Temporal Logics The previous de£nition introduces support sets in the context of boolean equation systems. At their lowest level many model checkers may be seen to manipulate such equation systems. However, conveying support-set-based information to users of model checkers requires the translation of boolean variables into user-level notations. The remainder of this section sketches how this can be done.

Users of model checkers typically input a (representation of) a Kripke structure and a formula in a temporal logic; the boolean variables used by the model checker represent assertions about whether or not a given state in the Kripke structure satis£es a given temporal formula derived from the formula input by the user. A *decorated support set* includes functions for extracting this information from boolean variables.

De£nition 4. *Let $T = \langle S, s_I, \rightarrow, V \rangle$ be a Kripke structure, ϕ be a formula in a temporal logic Φ (i.e. Φ is the set of formulas) with satisfaction relation $\models_T \subseteq S \times \Phi$. Also let $\mathcal{E}$ be a boolean equation system with $\mathcal{X} = lhs(\mathcal{E})$. Then $\langle \Gamma, \pi_T, \pi_\Phi \rangle$ is a* decorated support set *if $\Gamma = \langle r, X, \Xi \rangle$ is a support set over $\mathcal{X}$ and $\pi_T : \mathcal{X} \rightarrow S$, $\pi_\Phi : \mathcal{X} \rightarrow \Phi$ satisfy the following for all X_i such that $\Xi(X_i)$ is de£ned.*

1. *$\pi_S(X) = s_I$ and $\pi_\Phi(X) = \phi$.*
2. *If $\Xi(X_i)$ is de£ned then $\pi_S(X_i) \models_T \pi_\Phi(X_i)$ iff $r = \mathbf{1}$.*
3. *If $X_j \in \Xi(X_i)$ then either $\pi_T(X_i) = \pi_T(X_j)$ or $\pi_T(X_i) \rightarrow \pi_T(X_j)$.*

In a decorated support set, π_T and π_Φ extract state and temporal-formula information from the boolean variables in Γ. Condition 1 requires that the support variable of Γ be mapped to the start state of T and the initial formula ϕ, while Condition 2 stipulates that the value returned in Γ respect the semantics of the temporal logic. Condition 3 requires dependencies among boolean variables to "respect" T's transition relation.

4 Extracting Support Sets

As a generic vehicle for conveying model-checker reasoning, support sets are only useful to the extent that existing model checkers can be modi£ed to compute them. In this section we show how this may be done by presenting an extended example.

We begin by noting that for explicit-state mu-calculus model checkers, whether global [And94,CS93,EL86] or local [And94,BC96a,LRS98], the extraction of support sets is straightforward, since such procedures typically work by implicity or explicitly converting a mu-calculus model-checking problem into a boolean equation system as described in Section 2. For reasons of space we do not consider these further. Instead, in the remainder of this section we show how an automaton-based algorithm for CTL* may be modi£ed to construct support sets [KVW00]. This algorithm is not obviously mu-calculus-related; nevertheless, support-set information may be extracted without damaging the time or space complexity of the procedure.

In automaton-based model checking for CTL*, formulas are converted into tree automata accepting the trees that make the formula true. Checking whether a Kripke structure satis£es a formula involves determining whether or not the (in£nite) tree obtained by unwinding the Kripke structure is accepted by the formula's tree automaton. This acceptance check is typically performed by viewing the Kripke structure itself as a tree automaton accepting the (single) tree obtained by the unwinding process just mentioned, computing a product with it and the automaton for the formula in question, and then checking whether or not the resulting product automaton is nonempty.

The automaton-based model checker considered below comes from [KVW00], although for technical convenience the de£nitions of the automata used borrow ideas from [BCG01] as well. Recall that $\mathcal{A}$ is the (£xed) set of atomic propositions.

De£nition 5.
1. *An* alternating tableau transition system *is a tuple $\langle Q, \rightarrow, q_I, \ell\rangle$, where Q is a £nite set of states, $\rightarrow \subseteq Q \times Q$ is the transition relation, $q_I \in Q$ is the start state, and $\ell \in Q \rightarrow \mathcal{A} \cup \{\neg A \mid A \in \mathcal{A}\} \cup \{\wedge, \vee, [], \langle\rangle\}$ is the labeling, and the following holds for all $q \in Q$.*

$$|\{q' \mid q \rightarrow q'\}| \begin{cases} = 0 \text{ if } \ell(q) \in \mathcal{A} \cup \{\neg A \mid A \in \mathcal{A}\} \\ \geq 1 \text{ if } \ell(q) \in \{\wedge, \vee\} \\ = 1 \text{ if } \ell(q) \in \{[], \langle\rangle\} \end{cases}$$

2. *Alternating tableau transition system $\langle Q, \rightarrow, q_I, \ell\rangle$ is* hesitant *if for every nontrivial*[1] *strongly connected component $Q_i \subseteq Q$ of the graph $\langle Q, \rightarrow\rangle$ and every $q \in Q_i$, either $\ell(q) \in \{\wedge, []\}$ or $\ell(q) \in \{\vee, \langle\rangle\}$. In the former case Q_i is called* existential*, while in the latter it is called* universal*.*
3. *A* hesitant alternating tableau automaton (HATA) *is a tuple $\langle Q, \rightarrow, q_I, \ell, \langle G, B\rangle\rangle$ where $\langle Q, \rightarrow, q_I, \ell\rangle$ is a hesitant alternating tableau transition system and $G, B \subseteq Q$ constitute the acceptance condition.*

[1] A strongly connected component Q_i is nontrivial if there exist $q, q' \in Q_i$ such that $q \rightarrow q'$.

HATAs are very similar to the hesitant automata in [KVW00]; the only real difference is the use of labels on states rather than transitions to record "alternation information".

HATAs generate "runs" as they process Kripke structures.

De£nition 6. *Given HATA $M = \langle Q, \rightarrow, q_I, F, \ell, \langle G, B \rangle \rangle$ and a Kripke structure $T = \langle S, s_I, \rightarrow, V \rangle$, a* run *of M on T is a maximal tree in which the nodes are are labeled by elements of $S \times Q$ as follows. (1) The root of the tree is labeled by $\langle s_I, q_I \rangle$. (2) For each node σ labeled by $\langle s, q \rangle$:*

1. *If $\ell(q) \in A$ then σ is a leaf.*
2. *If $\ell(q) = \wedge$ and $\{q' \mid q \rightarrow q'\} = \{q_1, \cdots, q_m\}$, then σ has children $\sigma_1, \cdots, \sigma_m$, with σ_i labeled by $\langle s, q_i \rangle$.*
3. *If $\ell(q) = \vee$ then σ has one child, σ', which is lableled by $\langle s, q' \rangle$ for some $q' \in \{q' \mid q \rightarrow q'\}$.*
4. *If $\ell(q) = []$, $q \rightarrow q'$, and $\{s' | s \rightarrow s'\} = \{s_1, .., s_m\}$ then σ has children $\sigma_1, .., \sigma_m$, with σ_i is labeled by $\langle s_i, q' \rangle$.*
5. *If $\ell(q) = \langle\rangle$ and $q \rightarrow q'$ then σ has one child σ', and σ' is labeled by $\langle s', q' \rangle$ for some s' such that $s \rightarrow s'$.*

Note that any in£nite path in a run eventually consists only of states from the same nontrivial strongly connected component. We call such an in£nite path *existential* if this component is existential and *universal* otherwise. Because the transition relation of T is total, the only leaves in a run must be labeled either by $\langle s, A \rangle$ or $\langle s, \neg A \rangle$ for some $A \in \mathcal{A}$. We call leaves *successful* if they are labeled $\langle s, A \rangle$ and $s \in V(A)$ or $\langle s, \neg A \rangle$ and $s \notin V(A)$. A run is *successful* iff: every leaf is successful; every existential in£nite path contains in£nitely many occurrences of states in G; and every universal in£nite path contains £nitely many occurrences of states in B.

In order to determine whether or not a HATA M has a successful run on Kripke structure T, [KVW00] advocates checking the nonemptiness of a product automaton built from M and T. In the slightly revised setting considered here the nonemptiness check may be de£ned on an and-or graph $G_{T,M}$ de£ned as follows.

- The vertex set of $G_{T,M}$ is $S \times Q$.
- The edge relation E is de£ned by: $\langle \langle s, q \rangle, \langle s', q' \rangle \rangle \in E$ iff $\langle s', q' \rangle$ satis£es the conditions of being a child of $\langle s, q \rangle$ in some run of M on T.
- Labeling function $f \in S \times Q \rightarrow \{\wedge, \vee\}$ is de£ned as follows.

$$f(\langle s, q \rangle) = \begin{cases} \wedge \text{ if } \ell(q) \in \{\wedge, []\} \text{ or } \langle s, q \rangle \text{ would be a successful leaf in some run} \\ \vee \text{ otherwise} \end{cases}$$

The model-checking routine works by assigning truth values $\mathbf{0}$ and $\mathbf{1}$ to vertices in $G_{T,M}$. Details may be found in [KVW00], but the rough idea is to process the strongly connected components in $G_{T,M}$ in reverse topological order, starting with components containing no edges to other components. Strongly connected components in $G_{T,M}$ have the property that all nodes share the same label ($\wedge$ or $\vee$). Sink nodes in the graph correspond to successful or unsuccessful leaves in some run and are assigned $\mathbf{1}$ in the £rst case and $\mathbf{0}$ in the latter. The following process is then repeated for each component. First, values in lower components that the current component has edges into are "propagated upwards" into the current component, and new boolean values assigned to nodes in the current component in the obvious manner (i.e. a $\vee$-labeled node is assigned $\mathbf{1}$ if it has an edge to a node assigned boolean value $\mathbf{1}$, etc.). This propagation process is continued until no more is possible. If the component still has unlabeled nodes then all nodes are assigned $\mathbf{1}$ if the component is labeled $\vee$ and there exists a vertex in the component of form $\langle s, q\rangle$ for some $q \in G$, or if the component is labeled $\wedge$ and there is no vertex of form $\langle s, q\rangle$ for some $q \in B$; and $\mathbf{0}$ otherwise. It may be shown that $\langle s_I, q_I\rangle$ is assigned $\mathbf{1}$ if HATA M accepts Kripke structure T and $\mathbf{0}$ otherwise.

In order to extract support sets from the information computed by this model-checking algorithm, we £rst de£ne a boolean equation rom $G_{T,M}$ as follows. *Variables* correspond to vertices in $G_{T,M}$, while *right-hand sides* are constructed from labels and edges in $G_{T,M}$: if the label of a vertex v is $\wedge$, then the right-hand side for the variable v is $\bigwedge\{v' \mid \langle v, v'\rangle \in E\}$, and similary for $\vee$. *Blocks* are constructed from the strongly connected components of $G_{T,M}$. Let $G_1, \ldots, G_m$ be these components listed in topographical order: if $i < j$ then there is no edge from any node in G_j to G_i. For each G_i we construct two parity blocks E_i, E_i' as follows.

If G_i's label is $\vee$: Let E_i contain the equations whose left-hand sides $\langle s, q\rangle$ are nodes in G_i and with the property that $q \in G$. Let E_i' consist of the other equations whose left-hand sides are in G_i. Assign ν as the parity of E_i and μ as the parity of E_i'.

If G_i's label is $\wedge$: Let E_i contain the equations whose left-hand sides $\langle s, q\rangle$ are nodes in G_i and with the property that $q \in B$. Let E_i' consist of the other equations whose left-hand sides are in G_i. Assign μ as the parity of E_i and ν as the parity of E_i'.

The boolean equation system $\mathcal{E}_{T,M}$ is the sequence $E_1E_1' \ldots E_mE_m'$. One may prove the following.

Theorem 3. *Let T be a Kripke structure and M a HATA, and let s be a state in T and q a state in M. Then $[\![\mathcal{E}_{T,M}]\!](\langle s, q\rangle) = \mathbf{1}$ iff the value assigned to node $\langle s, q\rangle$ in $G_{T,M}$ by the model-checking algorithm of [KVW00] is $\mathbf{1}$.*

The evaluation procedure then computes the dependency set, denoted as $\xi(\langle s_k, q_k\rangle)$, for every vertex $\langle s_k, q_k\rangle$ in $G_{T,M}$. Recall that vertices in a strongly-connected component G_i are evaluated in two steps. Variable $\langle s_k, q_k\rangle$ is assigned a value in the £rst step if the values of its children determines permit; in this case we de£ne $\xi(\langle s_k, q_k\rangle)$ to contain the children whose value matches $\langle s_k, q_k\rangle$. In the second step, the remaining vertices in G_i are evaluated. We consider the case that G_i is existential; the case that G_i is universal can be handled similarly. If there is a vertex $\langle s', q'\rangle$ in G_i with $q' \in G$, then the other vertices in G_i have value $\mathbf{1}$. We build a spanning tree rooted at $\langle s', q'\rangle$ for the unassigned variables in G_i using the inverse edge relation E^{-1}. For each node $\langle s'', q''\rangle \neq \langle s', q'\rangle$ in the tree, we assign $\xi(\langle s'', q''\rangle)$ the singleton set containing the parent of $\langle s'', q''\rangle$ in the tree. We then make $\xi(\langle s', q'\rangle)$ contain one of its children in G_i with respect to E (the choice is abitrary). If there does not exist a $\langle s', q'\rangle$ such that $q' \in G$, then every remaining vertex $\langle s_k, q_k\rangle$ on G_i is assigned $\mathbf{0}$ and $\xi(s_k, q_k) = \{\langle s_l, q_l\rangle \mid \langle\langle s_k, q_k\rangle, \langle s_l, q_l\rangle\rangle \in E\}$. We now construct a support set $\Gamma = \langle r, \langle s, q\rangle, \Xi_{T,M}\rangle$ after $\langle s, q\rangle$ is labeled, where r is the label of $\langle s, q\rangle$, $\Xi(\langle s_k, q_k\rangle) = \xi(\langle s_k, q_k\rangle)$ if $\langle s_k, q_k\rangle$ is assigned r and $r = \mathbf{1}$, and $\Xi(\langle s_k, q_k\rangle) = \overline{\xi(\langle s_k, q_k\rangle)}$ if $\langle s_k, q_k\rangle$ is assigned r and $r = \mathbf{0}$ (this assignment in effect assigns the truth value $\mathbf{0}$ to every variable in $\xi(\langle s_k, q_k\rangle)$). One may check that Γ satis£es the requirements of being a support set and that "extracting" this support set does not affect the time or space complexity of the procedure.

We close this section with some comments about decorated support sets. In CTL* automaton-based model checking the HATA is constructed from a CTL* formula provided by the user. As the model checker should return a decorated support set, one may wonder how to de£ne the functions π_T and π_Φ. In the procedure just outlined the mapping π_T is straightforward, since every boolean variable corresponds to a pair $\langle s, q\rangle$, where s is a system state. As for π_ϕ, the HATA constructions in [BCG01,KVW00] work by associating HATA states with (sets of) CTL* propositions. These CTL* propositions can then be returned by π_Φ.

5 Diagnostic Information

In the remainder of the paper we study two different applications for support sets. In this section we show how support sets may be used to compute *diagnostic information* in general, and *linear witnesses* in particular. We note that support sets may also be used to compute winning strategies for the purposes of diagnostic routines based on game-based model checking [SS98], although we do not pursue this point here.

Counterexamples are used to indicate why a Kripke structure fails to satisfy a temporal property. Intuitively, a counterexample is a part of system "responsible" for the property being violated. Dually, when system satis£es a temporal property, a user may still desire some explanation given as a portion of the system, called a *witness*, responsible for the property being satis£ed. Although counterexample generators have existed for a number of years, to the best of our knowledge [CGMZ95] represents the £rst systematic explanation of how they work. Their de£nitions in the setting of CTL require that counterexamples and witnesses to be *linear*, i.e., execution paths of the system. In general the existence of linear counterexamples / witnesses depends on the structure of formulas as well as the Kripke structure being checked. In the case of CTL, for example, linear counterexamples (witnesses) exist if the primary path quanti£er used is A (E). [KV99] gives more general conditions for CTL* and shows show that judging whether a CTL* formulae admits such counterexamples / witnesses is PSPACE-complete.

Here we show how support sets may be used to generate linear counterexamples / witnesses without reference to the temporal logic in which system properties are formulated. In the rest of this section we restrict our attention to Kripke structures that are *self-loop-free*: no state s has the property that $s \rightarrow s$.

De£nition 7. *Support set $\langle r, X, \Xi\rangle$ is* linear *if for all X_i such that $\Xi(X_i)$ is de£ned, $|\Xi(X_i)| \leq 1$. Decorated support set $\langle \Gamma, \pi_T, \pi_\Phi\rangle$ is linear if Γ is.*

If a decorated support set is linear then one may extract a *linear witness* to the result contained in the support set as follows. Let $\Gamma = \langle \Gamma' = \langle r, X, \Xi\rangle, \pi_T, \pi_\Phi\rangle$ be a linear decorated support set for Kripke structure T. Then the *state projection* $\pi_S(\Gamma')$ is de£ned as follows: let $X_1 X_2 \ldots X_n$ be a depth-£rst search of the graph induced by Ξ beginning at $X = X_1$. Then $\pi_S(\Gamma) = \pi_T(X_1) \ldots \pi_T(X_n)$. In general, $\pi_S(\Gamma)$ is not an execution sequence of T, since the de£nition of decorated support set allows the states associated with adjecent variables in Γ to be the same, and hence not connected by a transition. However, a subsequence of $\pi(\Gamma)$ is guaranteed to be a computation path of T: delete all but one occurrence of a state in contigous subsequences containing only this state. Let $\pi(\Gamma)$ be this sequence; it is easy to show that it is a computation path in T that is a linear model of the result reported by the model checker.

In general support sets are not linear, but they can *minimized* in the following sense. A support set $\Gamma = \langle r, X, \Xi\rangle$ is *minimal* if the following conditions hold.

1. For every X' such that $\Xi(X')$ is de£ned, $X \underset{\Gamma}{\rightarrow}^{*} X'$ (i.e. X' affects X).
2. If $r = \mathbf{1}$ then for every variable X' whose right-hand side uses $\bigvee$ such that $\Xi(X')$ is de£ned, $|\Xi(X')| = 1$, and dually for $r = \mathbf{0}$.

Intuitively, a support set is minimal if it contains no extraneous information. It is straightforward to convert a support set $\Gamma = \langle r, X, \Xi\rangle$ into a minimal support set $\Gamma_{\min} =$

$\langle r, X, \Xi_{\min} \rangle$. The witness-extraction procedure can be applied to support sets that, while not linear themselves, minimize to linear support sets. Finally, we note that even when minimial support sets are not linear, they may be used to generate "recursive" linear witnesses *Á la* [CGMZ95] when all but one element in $\Xi(X')$ are guaranteed to belong to different strongly connected components than X' for any X'.

6 Certifying Model-Checking Results

In this section we give an ef£cient algorithm to check the validity of a support set submitted by a model checker. Such a routine has several practical motivations [Nam01]:

- It can be used to check for bugs in model checkers: if a support set returned by a checker is in fact not a support set, then the checker's reasoning is faulty.
- Support sets can be used as "certi£cates" for system correctness. A validity checker can then be used to check the "internal consistency" of such a certi£cate.

In what follows we £x boolean equation system $\mathcal{E} = E_1 \ldots E_n$. Let $\Gamma = \langle r, X, \Xi \rangle$ be a support set for $\mathcal{E}$ submitted by a checker. Without loss of generality, assume $r = \mathbf{1}$. Validating Γ amounts to checking that Properties I, II and III in De£nition 3 hold. Properties I and II Γ can be easily ascertained with routines that execute in $O(|\Gamma|)$. Checking Property III on Γ can be reduced to an *even-cycle* problem on labeled directed graphs. A labeled directed graph is a tuple $G = \langle D, V, E, \ell \rangle$, where $\ell : V \rightarrow D$ labels each vertex with a element from D. The *even-cycle* problem is given as follows: given a labeled directed graph $G = \langle \{1, 2, \cdots, k\}, V, E, \ell \rangle$, determine whether there is a cycle ρ in it such that $min_{v \in \rho}\{\ell(v)\}$ is even.

Γ induces a labeled directed graph $G = \langle \{1, 2, \cdots, k\}, V, E, \ell \rangle$ as follows. V is the set of all variables de£ned on Ξ, E is the relation $\underset{\Gamma}{\rightarrow}$. ℓ satis£es the following criteria.

- If $X', X'' \in lhs(E_i)$ then $\ell(X') = \ell(X'')$.
- If $i < j$ then $\ell(E_i) \leq \ell(E_j)$. (Here $\ell(E_i)$ is the common value shared by all left-hand sides in E_i.)
- If the parity of E_i is μ then $\ell(E_i)$ is even; otherwise, $\ell(E_i)$ is odd.

A labeling satisfying these properties can easily be constructed in $|\mathcal{E}|$ time with $k \leq n$, where n is the number of blocks. Checking III on Γ is equivalent to checking whether there is an even cycle in G. [KKV01] shows that the even-cycle problem can be solved in $O((|V| + |E|)log(\lceil \frac{k}{2} \rceil))$. Their approach is a variant of an algorithm for hierarchical clustering [Tar82]. As our construction of ℓ above restricts $k \leq n$, checking property III can be done in $O(|\Xi| log(\lceil \frac{n}{2} \rceil))$, where n is the number of parity blocks of $\mathcal{E}$.

The complexity can be improved by noticing that an even-cycle can only exist in a strongly-connected component of G_Γ. Therefore, we can check each strongly-connected component independently. By De£nition 2, the maximal labeling number k won't exceed the alternation depth of $\mathcal{E}$. Thus, the overall time complexity is $O(|\mathcal{E}| \cdot |\mathcal{T}| \cdot log(\lceil \frac{ad(\mathcal{E})}{2} \rceil))$, which is less than the lower bound of μ-calculus model-checking. This suggests that the certi£er will not in general increase the cost of overall complexities of a veri£cation tool.

7 Conclusions and Related Work

In this paper we have presented support sets as a generic data structure for conveying "meta-model-checking" results, i.e. results regarding the means by which model-checking answers are arrived at. We showed how model checkers may be modi£ed to return support sets and how support sets may be used to generate diagnostic information and may be ef£ciently checked for internal consistency. We have also studied other uses for support sets not mentioned in this paper, including vacuity checking [KV99]. Prototype implementations of these results are being investigated in the context of the CWB-NC veri£cation tool [CS96].

The idea of retaining evidence during model checking as a basis for justifying the result has appeared in several recent publications. In [PZ01,PPZ01] ideas in the setting of linear-time temporal-logic are presented. Regarding the mu-calculus, [Mat00] uses a distinguised solution to alternation-free boolean equation system, called *extended boolean graphs* (EBGs), to encode the proof structures. EBGs can be viewed as a special case of support set in the alternation-free fragment of the mu-calculus. Even closer to this work is that in [Nam01], which uses deductive proofs to encode evidence for model-checking in the modal mu-calculus. That paper also discusses some of the same applications mentioned here for deductive information; a technical point of departure, however, is that deductive proofs in that setting require extra information in form of *ranking information* which records information on the number of "approximations" of outer variables that an inner variable depends on. This requirement plays the same role as the circularity restriction for support sets: in fact, the ranking information speci£es the position of a variable in a dependency loop. With this extra information verifying the validity of proofs is easier than the veri£cation for support sets. An obvious drawback is that storing ranking information requires additional space, and it also requires model checkers to maintain the information about numbers of approximations for variables. This information is not typically computed by on-the-¤y (local) algorithms due to its top-down evaluation fashion. Therefore, it is not clear how ranking information can be collected for local algorithms. On the other hand, support sets require only dependency information, which is computed by both global and local algorithms.

References

[And94] H.R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[BBDER97] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Ef£cient detection of vacuity in ACTL formulas. In *Proceedings of the Ninth International Conference on Computer Aided Veri£cation (CAV '97), LNCS 1254*. Springer-Verlag, 1997.

[BC96a] G. S. Bhat and R. Cleaveland. Ef£cient local model checking for fragments of the modal μ-calculus. In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), LNCS 1055*. Springer-Verlag, March 1996.

[BC96b] G. S. Bhat and R. Cleaveland. Ef£cient model checking via the equational μ-calculus. In E. M. Clarke, editor, *11th Annual Symposium on Logic in Computer Science (LICS '96)*, pages 304–312, New Brunswick, NJ, July 1996. Computer Society Press.

[BCG01] G. Bhat, R. Cleaveland, and A. Groce. Ef£cient model checking via büchi tableau automaton. In *Proceedings of the Seventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01), LNCS 2031*. Springer-Verlag, 2001.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic veri£cation of £nite-state concurrent systems using temporal logic speci£cations. *ACM TOPLAS*, 8(2), 1986.

[CGMZ95] E. Clarke, O. Grumberg, K. McMillian, and X. Zhao. Ef£cent generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd Design Automaton Conference*, San Francisco, CA, 1995.

[CKV01] H. Chockler, O. Kupferman, and M. Vardi. Coverage metrics for temporal logic model checking. In *Proceedings of the Seventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01), LNCS 2031*. Springer-Verlag, April 2001.

[CS93] R. Cleaveland and B. U. Steffen. A linear-time model checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[CS96] R. Cleaveland and S. Sims. The NCSU concurrency workbench. In R. Alur and T. A. Henzinger, editors, *Computer Aided Veri£cation (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, New Brunswick, New Jersey, July 1996. Springer-Verlag.

[Dam94] M. Dam. CTL* and ECTL* as fragement of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[EL86] E.A. Emerson and C.-L. Lei. Ef£cient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science (LICS '86)*, pages 267–278, Cambridge, Massachusetts, June 1986. Computer Society Press.

[KKV01] V. King, O. Kupferman, and M. Y. Vardi. On the complexity of parity word automaton. In *The 4th International Conference on Foundations of Software Science and Computation Structures,* Vol. 2030 of *Lecture Notes in Computer Science*, 2001.

[KV99] O. Kupferman and M. Y. Vardi. Vacuity detection in temporal model checking. In *Proceedings of the Tenth Conference on Correct Hardware Design and Veri£cation Mothods, LNCS 1703*, 1999.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[LRS98] X. Liu, C. R. Ramakrishnan, and S. A. Smolka. Fully local and ef£cient evaluation of alternating £xed points. In *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98), LNCS 1389*. Springer-Verlag, 1998.

[Mad97] A. Mader. *Veri£cation of Modal Properties Using Boolean Equation Systems*. PhD thesis, Müchen, Techn-Univ., 1997.

[Mat00] R. Mateescu. Ef£cient diagnostic generation for boolean equation system. In *Proceedings of the Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '00),* Vol. 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2000.

[Nam01] K. Namjoshi. Certifying model checkers. In *Proceedings of the 13th International Conference on Computer Aided Veri£cation (CAV '01), LNCS 2102*. Springer-Verlag, 2001.

[PPZ01] D. Peled, A. Pnueli, and L. Zuck. From falsi£cation to veri£cation. In *FST&TCS*, volume 2245 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[PZ01] D. Peled and L. Zuck. From model checking to a temporal proof. In M. Dwyer, editor, *SPIN 2001*, volume 2057 of *Lecture Notes in Computer Science*, pages 1–14, Toronto, May 2001. Springer-Verlag.

[QS82] J. P. Queille and J. Sifakis. Speci£cation and veri£cation of concurrent systems in Cesar. In *Proceedings of the International Symposium in Programming*, volume 137 of *Lecture Notes in Computer Science*, Berlin, 1982. Springer-Verlag.

[SS98] P. Stevens and C. Stirling. Practical model checking using games. In *Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98), LNCS 1389*. Springer-Verlag, 1998.

[Sti95] C. Stirling. Local model checking games. In I. Lee and S. A. Smolka, editors, *Proceedings of the Sixth International Conference on Concurrency Theory (CONCUR '95),* Vol. 962 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Tar82] R. E. Tarjan. A hierarchical clusting algorithm using strong components. *Information Processing Letters*, 14:26–29, 1982.