Automata as Abstractions

Dennis Dams and Kedar S. Namjoshi

Bell Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.
{dennis,kedar}@research.bell-labs.com

Abstract. We propose the use of tree automata as abstractions in the verification of branching time properties, and show several benefits. In this setting, soundness and completeness are trivial. It unifies the abundance of frameworks in the literature, and clarifies the role of concepts therein in terms of the well-studied field of automata theory. Moreover, using automata as models simplifies and generalizes results on maximal model theorems.

1 Introduction

Program verification, and in particular the model checking [3, 27] approach that we consider here, usually takes the form of property checking: Given a program model M and a property φ, does M satisfy φ ($M \models \varphi$)? The answer obtained should be *true* or *false*; otherwise verification has failed. *Program analysis* [26], on the other hand, serves a somewhat different purpose, namely to collect information about a program. Thus, program analysis produces a set of properties that M satisfies. The more properties there are, the better: this enables more compiler optimizations, better diagnostic messages, etc.

Abstraction is fundamental to both verification and analysis. It extends model checking to programs with large state spaces, and program analyses can be described in a unified way in terms of Abstract Interpretation [5]. An *abstraction framework* includes the following components. The set C of *concrete objects* contains the structures whose properties we are principally interested in, such as programs. A is the set of *abstract objects* (or *abstractions*), which simplify concrete objects by ignoring aspects that are irrelevant to the properties to be checked (in verification) or collected (in analysis), thus rendering them amenable to automated techniques. An *abstraction relation* $\rho \subseteq C \times A$ specifies how each concrete object can be abstracted. Properties are expressed in a *logic* L and interpreted over concrete objects with $\models \subseteq C \times L$ and over abstract objects[1] with $\models^\alpha \subseteq A \times L$. A principal requirement for any abstraction framework is that it is *sound*: if $\rho(c, a)$ and $a \models^\alpha \varphi$, then $c \models \varphi$. This ensures that we can establish properties of concrete objects by inspecting suitable abstractions.

[1] One could choose different logics on the concrete and abstract sides, but this would unnecessarily complicate the discussion here.

Abstraction for analysis and verification In program analysis, depending on the kind of information that needs to be collected, a particular *abstract data domain* is chosen that provides *descriptions* of concrete data values. The abstract object is then, e.g., a non-standard collecting semantics, computed by "lifting" all program operations to the abstract domain. Soundness is ensured by showing that each lifted operation correctly mimics the effect of the corresponding concrete operation. Ideally, lifted operations are *optimal*, which means that the largest possible set of properties is computed relative to the chosen abstract domain. As this is not always possible, the *precision* of a lifted operation is of interest.

In Model Checking, the properties of interest go beyond the universal safety properties that are commonly the target of program analyses; they include liveness aspects and existential quantification over computations, as formalized by branching-time temporal logics. Under these circumstances, the usual recipe for lifting program transitions to abstract domains falls short: the abstract programs thus constructed are sound only for universal, but not for existential properties. This can be fixed by lifting a transition relation in two different ways, interpreting universal properties over one relation (called *may*), and existential ones over the other relation (called *must*) [22].

Given an abstract domain, optimal *may* and *must* relations can be defined [4, 7, 28]. But one may argue that for program verification, the notion of precision is overshadowed by the issue of choosing a suitable abstract domain. In verification, unlike in analysis, a partial answer is not acceptable: one wants to either prove or disprove a property. Hence, even an optimal abstraction on a given domain is useless if it does not help settle the verification question. In other words, the focus shifts from precision of operators to precision of domains. Tools for verification via abstraction will need to be able to construct modal transition systems over domains of varying precision, depending on the given program and property.

Is it, then, enough to consider a may-must transition system structure over arbitrarily precise abstract domains? *No*, suggest a number of research results [23, 25, 30, 6, 10]: modifying must transitions so as to allow multiple target states — a *must hyper-transition* — enables one to devise even more precise abstract objects, which satisfy more existential properties. Are there other missing ingredients? When have we added enough? To answer these questions, we first need to formulate a reasonable notion of "enough".

From precision to completeness As we have argued earlier, precision is not really the key abstraction issue in the context of verification. Even within the limited setting where abstract objects are finite transition systems with only *may* transitions, it is always possible to render more universal properties true, by making a domain refinement. The implicit question in the above-mentioned papers is a different one, namely: is it always possible to find a *finite* abstract object that is precise enough to prove a property true of the concrete object? (The emphasis on finiteness is because the end goal is to apply model checking to the abstract object.) This is the issue of (in)completeness: An abstraction framework is *com-*

plete[2] if for every concrete object $c \in C$ and every property $\varphi \in L$ such that $c \models \varphi$, there exists a *finite* abstract object $a \in A$ such that $\rho(c, a)$ and $a \models \varphi$. For the case of linear-time properties, completeness was first addressed in [32, 20].

The quest for completeness for branching time, undertaken in [25, 6], has shown that without the addition of *must* hyper-transitions (obtained by so-called *focus* moves in [6]), modal transition systems are incomplete for existential safety properties. Furthermore, as already predicted by [32, 20], *fairness conditions* are needed to achieve completeness for liveness properties.

Contribution of this paper Over the years, research in refinement and abstraction techniques for branching time properties on transition systems has produced a large and rather bewildering variety of structures: *Modal Transition Systems*, with *may* and *must* relations [22]; *Abstract Kripke structures* [9] and *partial* and *multi-valued Kripke structures* [1, 2], with 3-valued components; *Disjunctive Modal Transition Systems* [23], *Abstract transition structures* [10], and *Generalized Kripke Modal Transition Systems* [30], with *must* hyper-transitions; and *Focused Transition Systems*, with focus and defocus moves and acceptance conditions [6]. Having achieved completeness for full branching time logic with Focused Transition Systems, which put all essential features together, it may be time to step back to try and see the bigger picture in this abundance of concepts. Is there an encompassing notion in terms of which the key features of all of these can be understood?

In this paper we answer this question affirmatively: indeed, behind the various disguises lives the familiar face of *tree automata*. We start by showing how automata themselves can be employed as abstract objects, giving rise to remarkably simple soundness and completeness arguments. The technical development rests upon known results from automata theory. We view this as a strong positive: it shows that establishing the connection enables one to apply results from the well-developed field of automata theory in a theory of abstraction. Then, we connect automata to previously proposed notions of abstract objects, by showing how one of the existing frameworks can be embedded into the automaton framework.

A tree automaton, indeed, can be seen as just an ordinary (fair) transition system extended with an OR-choice (or "focus") capability. Remarkably, this simple extension turns out to be enough to guarantee completeness. Automata thus identify a minimal basis for a complete framework, showing that some of the concepts developed in modal/mixed/focused structures are not strictly necessary[3]. As a further illustration of the clean-up job achieved by our observation,

[2] A different notion of completeness is studied by Giacobazzi et. al. in [14]. Their notion of completeness requires that for every concrete object, there exists an abstraction of it that satisfies precisely all the properties as the concrete object, relative to a given logic. For example, in the context of CTL, this requires the concrete and abstract transition systems to be bisimilar, and thus there is not always a finite abstraction.

[3] This conclusion applies only to the issue of completeness: in terms of size, for instance, focused transition systems may be exponentially more compact than ordinary

we illustrate how the use of automata generalizes and simplifies known *maximal model theorems* [15, 21].

An appetizer To demonstrate how the use of automata as abstractions simplifies matters, we consider the notions of soundness and completeness in an abstraction framework that uses automata (details follow in subsequent sections). For an automaton $\mathcal{A}$ considered as an abstract object, the set of concrete objects that it abstracts (its *concretization*) is taken to be its (tree[4]) language $\mathcal{L}(\mathcal{A})$.

The question is how to define the evaluation of temporal properties over tree automata such that soundness is ensured. Adopting the automata-theoretic view, we express also a property φ by an automaton, whose language consists of all models of φ. Clearly, the answer then is to define, for any automaton $\mathcal{A}$ and property $\mathcal{B}$, $\mathcal{A} \models \mathcal{B}$ as $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. Soundness then holds trivially: if $M \in \mathcal{L}(\mathcal{A})$ and $\mathcal{A} \models \mathcal{B}$, then $M \in \mathcal{L}(\mathcal{B})$. Furthermore, also completeness follows immediately: Given any, possibly infinite, M such that $M \models \mathcal{B}$, there exists a finite abstraction of M through which $\mathcal{B}$ can be demonstrated, namely $\mathcal{B}$ itself: clearly, $M \in \mathcal{L}(\mathcal{B})$ and $\mathcal{B} \models \mathcal{B}$. All this is trivial, and that is precisely the point: using automata, constructions that are otherwise rather involved now become straightforward.

In practice, the above set-up is less appealing, since checking a property over an abstraction requires deciding tree-language inclusion, which is EXPTIME-hard. In Section 3, we define a notion of simulation between tree automata. Deciding the existence of such a simulation has a lower complexity (in NP, and polynomial in the common case), yet it is a sufficient condition for language inclusion. We show that the approach remains sound and complete for this choice.

2 Background

In the introduction, we make an informal case that tree automata are more appropriate than transition systems as the objects of abstraction. Tree automata are usually defined (cf. [12]) over complete trees with binary, ordered branching (i.e., each node has a 0-successor and a 1-successor). This does not quite match with branching time logics: for example, the basic EX operator of the μ-calculus cannot distinguish between the order of successors, or between bisimilar nodes with different numbers of successors. In [18, 19], Janin and Walukiewicz introduced a tree automaton type appropriately matched to the μ-calculus, calling it a *μ-automaton*. We use this automaton type in the paper.

Definition 1 (Transition System, Kripke Structure). *A* transition system with state labels from *Lab is a tuple* $\mathcal{S} = (S, \hat{S}, R, L)$ *where* S *is a nonempty,*

automata, since they can exploit both 3-valuedness (in propositional labelings and in transitions) and alternation.

[4] We focus on verification of branching-time properties, and consequently use tree automata as abstractions. But our suggestion to use automata as abstractions specializes to the case of linear-time properties and word automata, and indeed was inspired by it — see [20].

countable set of states, $\hat{S} \subseteq S$ *is a set of* initial states, $R \subseteq S \times S$ *is a* transition relation, *and* $L : S \to Lab$ *is a* labeling function.

Fix $Prop$ *to be a finite set of propositions. A* Kripke Structure *is a transition system with state labels from* 2^{Prop}. □

From each initial state, a transition system can be "unfolded" into its computation tree. Formally, a *tree* is a transition system with state space isomorphic to a subset of strings over the naturals such that if $x.c$ is a state, so is x, and there is a transition from x to $x.c$. The state corresponding to the empty string is called the *root*. We refer to a state in a tree as a *node*.

Definition 2 (μ-Automaton [18]). *A* μ-automaton[5] *is a tuple* $\mathcal{A} = (Q, B, \hat{Q}, \text{OR}, \text{BR}, L, \Omega)$ *where:*

- Q *is a non-empty, countable set of states, called* OR *states,*
- B *is a countable set of states, disjoint from* Q*, called* BRANCH *states,*
- $\hat{Q} \subseteq Q$ *is a non-empty set of* initial states*,*
- $\text{OR} \subseteq Q \times B$ *is a* choice relation*, from* OR *states to* BRANCH *states,*
- $\text{BR} \subseteq B \times Q$ *is a* transition relation*, from* BRANCH *states to* OR *states,*
- $L : B \to 2^{Prop}$ *is a* labeling function*, mapping each* BRANCH *state to a subset of propositions,*
- $\Omega : Q \to \mathbb{N}$ *is an* indexing function*, used to define the* acceptance condition.

□

We sometimes Curry relations: for instance, $\text{OR}(q)$ is the set $\{b \mid (q, b) \in \text{OR}\}$. The automaton is *finite* iff $Q \cup B$ is a finite set. Only finite automata are formulated in [18]; we allow automata to be infinite so that an infinite transition system can be viewed as a simple tree automaton. (Indeed, μ-automata generalize fair transition systems only in allowing non-trivial OR choice relations. This is made precise in Definition 5 of the next section.) In the rest of the paper, we use "automaton" to stand for "μ-automaton", unless mentioned otherwise.

Informal Semantics: Given an infinite tree, a run of an automaton on it proceeds as follows. The root of the tree is tagged with an initial automaton state; a pair consisting of a tree node and an automaton state is called a *configuration*. At a configuration (n, q), the automaton has several choices as given by $\text{OR}(q)$; it chooses (non-deterministically) a BRANCH state b in $\text{OR}(q)$ whose labeling matches that of n. The automaton tags the children of n with OR states in $\text{BR}(b)$, such that every OR-state tags some child, and every child is tagged with some OR state. This results in a number of successor configurations, which are explored in turn, ad infinitum. Notice that there can be many runs of an automaton on a tree, based on the non-determinism in choosing BRANCH states in the automaton, and in the way children are tagged in the tree. An input tree is accepted if there is *some* run where every sequence of configurations produced

[5] We have made some minor syntactic changes over the definition in [18]: making the role of the BRANCH states explicit, allowing multiple initial states, and eliminating transition labels.

on that run meets the automaton acceptance condition. To illustrate this process, Figure 1 shows a tree, an automaton for the CTL formula EFp ("a state labeled p is reachable"), and an accepting run of the automaton on the tree.

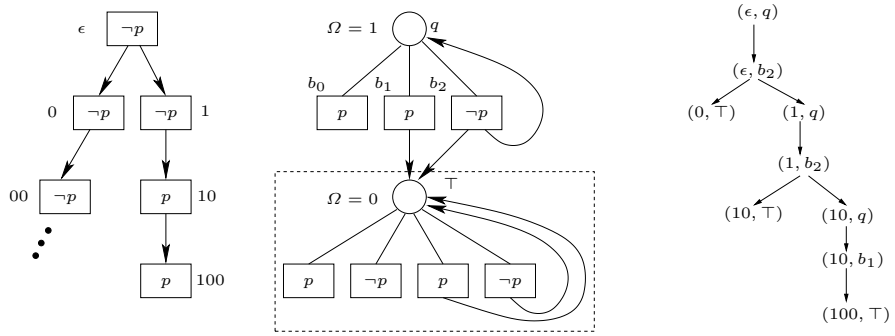

Fig. 1. Left: an input tree. Middle: μ-automaton for EFp, taking $Prop = \{p\}$. The state $\top$ accepts any subtree. Right: an accepting run.

Definition 3 (Automaton Acceptance [18]). *Let $\mathcal{S} = (S, \{\epsilon\}, R, L_{\mathcal{S}})$ be a tree with labels from 2^{Prop}, and let $\mathcal{A} = (Q, B, \hat{Q}, \text{OR}, \text{BR}, L_{\mathcal{A}}, \Omega)$ be an automaton. For $\hat{q} \in \hat{Q}$, a* $\hat{q}$-run of $\mathcal{A}$ on $\mathcal{S}$ *is a tree $\mathcal{T}$ where each node is labeled with a configuration from $S \times (Q \cup B)$, satisfying the following conditions.*

1. *(Initial) The root of $\mathcal{T}$ is labeled with $(\epsilon, \hat{q})$.*
2. *(*OR*) Every node of $\mathcal{T}$ that is labeled with (n, q), where $q \in Q$, has a child labeled (n, b) for some $b \in \text{OR}(q)$.*
3. *(*BRANCH*) For every node $x \in \mathcal{T}$ that is labeled with (n, b) where $b \in B$:*
   (a) *$L_{\mathcal{S}}(n) = L_{\mathcal{A}}(b)$.*
   (b) *For every $n' \in R(n)$, there is a child of x labeled with (n', q'), for some $q' \in \text{BR}(b)$.*
   (c) *For every $q' \in \text{BR}(b)$, there is a child of x labeled with (n', q'), for some $n' \in R(n)$.*

A $\hat{q}$-run $\mathcal{T}$ of $\mathcal{A}$ on $\mathcal{S}$ is accepting *(by the so-called "parity condition") iff on every infinite path π in $\mathcal{T}$, the least value of $\Omega(q)$, for* OR*-states q that appear infinitely often on π, is* even. *The tree $\mathcal{S}$ is* accepted *by $\mathcal{A}$ iff for some $\hat{q} \in \hat{Q}$, there is a $\hat{q}$-run of $\mathcal{A}$ on $\mathcal{S}$ that is accepting. A Kripke Structure is accepted by $\mathcal{A}$ iff all trees in its unfolding are accepted by $\mathcal{A}$. The* language $\pounds(\mathcal{A})$ of $\mathcal{A}$ *is the set of all Kripke Structures that are accepted by $\mathcal{A}$.* □

3 Abstraction with Automata

3.1 Abstraction with Language Inclusion

We now define the *abstraction framework based on automaton language inclusion* that was discussed at the end of the introduction. The concrete objects are Kripke Structures. The abstract objects are *finite* automata. The abstraction relation is language membership: i.e., a Kripke Structure $\mathcal{S}$ is abstracted by automaton $\mathcal{A}$ iff $\mathcal{S} \in \mathcal{L}(\mathcal{A})$. Finally, branching time temporal properties are given as *finite* automata, where a property $\mathcal{B}$ holds of a Kripke Structure $\mathcal{S}$ (i.e., a concrete object) iff $\mathcal{S} \in \mathcal{L}(\mathcal{B})$, and $\mathcal{B}$ holds of an automaton $\mathcal{A}$ (i.e., an abstract object) iff $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. In the introduction we showed the following.

Theorem 1. *The abstraction framework based on automaton language inclusion is sound and complete.*

3.2 Abstraction with Simulation

The simplicity of the framework presented above makes it attractive from a conceptual point of view. However, checking a temporal property amounts to deciding language inclusion between tree automata, which is quite expensive (EXPTIME-hard even for the finite tree case [29]). Hence, we consider below a framework based on a sufficient condition for language inclusion, namely the existence of a simulation between automata, which can be checked more efficiently.

For automata on finite trees, simulation has been defined previously, see e.g. [8], and our definition here is a straightforward generalization of that. Roughly speaking, simulation between automata $\mathcal{A}_1$ and $\mathcal{A}_2$ ensures that at corresponding OR states, any OR choice in $\mathcal{A}_1$ can be simulated by an OR choice in $\mathcal{A}_2$. As such, it follows the structure of the standard definition of simulation between transition systems [24]. At corresponding BRANCH states, however, the requirement is more reminiscent of the notion of *bisimulation*: any BRANCH transition from one automaton has a matching BRANCH transition from the other. In order to deal with the infinitary acceptance conditions, it is convenient to describe simulation checking as an infinite, two-player, game, as is done in [16] for fair simulation on Kripke Structures.

Definition 4 (Automaton Simulation). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be automata. For initial states $\hat{q}_1 \in \hat{Q}_1$ and $\hat{q}_2 \in \hat{Q}_2$, we define the $(\hat{q}_1, \hat{q}_2)$-*game *as follows. Every play is a sequence of configurations as specified by the following rules. Each configuration consists of a pair of states of the same type (i.e., both are* OR *states or both are* BRANCH *states).*

1. *(Initial) The initial configuration is $(\hat{q}_1, \hat{q}_2)$.*
2. *(*OR*) In an "*OR*" configuration (q_1, q_2) (where $q_1 \in Q_1$ and $q_2 \in Q_2$), Player II chooses b_1 in $\text{OR}(q_1)$; Player I has to respond with some b_2 in $\text{OR}(q_2)$, and the play continues from configuration (b_1, b_2).*
3. *(*BRANCH*) In a "*BRANCH*" configuration (b_1, b_2) (where $b_1 \in B_1$ and $b_2 \in B_2$), each of the following are continuations of the play:*

(a) *(Prop) In this continuation, the play ends and is a win for Player I if* $L_1(b_1) = L_2(b_2)$*, and it is a win for Player II otherwise.*
(b) *(Bisim) Player II chooses a 'side'* i *in* $\{1,2\}$*, and an* OR*-state* q_i *in* $\text{BR}_i(b_i)$*; Player I must respond with an* OR*-state* q_j *in* $\text{BR}_j(b_j)$*, from the other side* j *(i.e.,* $j \in \{1,2\}; j \neq i$*) and the play continues from configuration* (q_1, q_2)*.*

If a finite play ends by rule 3a, the winner is as specified in that rule. For an infinite play π*, and* $i \in \{1,2\}$*, let* $proj_i(\pi)$ *be the infinite sequence from* Q_i^ω *obtained by projecting the* OR *configurations of* π *onto component* i*. Then* π *is a win for Player I iff either* $proj_1(\pi)$ *does not satisfy the acceptance condition for* $\mathcal{A}_1$*, or* $proj_2(\pi)$ *satisfies the acceptance condition for* $\mathcal{A}_2$*.*

We say that $\mathcal{A}_1$ is simulated by $\mathcal{A}_2$*, written* $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$*, if for every* $\hat{q}_1 \in \hat{Q}_1$*, there exists* $\hat{q}_2 \in \hat{Q}_2$ *such that player I has a winning strategy for the* $(\hat{q}_1, \hat{q}_2)$*-game.* □

Theorem 2. *If* $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ *then* $\pounds(\mathcal{A}_1) \subseteq \pounds(\mathcal{A}_2)$*.*

Theorem 3. *Deciding the existence of a simulation relation between finite automata is in NP, and can be done by a deterministic algorithm that is polynomial in the size of the automata and exponential in the number of parity classes.*

Proof. (sketch) A winning strategy for Player I in the simulation game corresponds to a tree labeled with configurations where every path satisfies the winning condition. It is easy to construct a finite automaton that accepts such trees. The automaton remembers the current configuration and which player's turn it is, while the transitions of the automaton ensure that the successors in the tree are labeled with configurations that respect the constraints of the game (e.g., a node where player II takes a turn must have all possible successor configurations for a move by player II). This automaton is of size proportional to the product of the original automaton sizes. Its acceptance condition is that of the game. A parity condition can be written as either a Rabin or a Streett (complemented Rabin) condition, so the winning condition for the game, which has the shape $(\neg(\mathit{parity}) \vee \mathit{parity})$, is a Rabin condition. Thus, the existence of a winning strategy reduces to the non-emptiness of a non-deterministic Rabin tree automaton. The complexity results then follow from the bounds given in [11] for this question. If the acceptance conditions are Büchi, the simulation check is in polynomial time.

The *simulation-based framework* is defined like the one based on language inclusion, except that a branching-time temporal property $\mathcal{B}$ is defined to hold of an automaton $\mathcal{A}$ (i.e., an abstract object) iff $\mathcal{A} \sqsubseteq \mathcal{B}$. Soundness and completeness are again easy to show.

Theorem 4. *The simulation based framework is sound and complete.*

Proof. (Soundness) Let $\mathcal{S}$ be a Kripke structure, and $\mathcal{B}$ an automaton property. Suppose that $\mathcal{A}$ is an abstraction of $\mathcal{S}$ $(\mathcal{S} \in \pounds(\mathcal{A}))$ which satisfies property

$\mathcal{B}$ ($\mathcal{A} \sqsubseteq \mathcal{B}$). By Theorem 2, it follows that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. So it follows that $\mathcal{S} \in \mathcal{L}(\mathcal{B})$, i.e. $\mathcal{S}$ satisfies property $\mathcal{B}$.

(Completeness) Let $\mathcal{S}$ be a Kripke Structure that satisfies an automaton property $\mathcal{A}$ ($\mathcal{S} \in \mathcal{L}(\mathcal{A})$). So $\mathcal{A}$ itself is an abstraction of $\mathcal{S}$. Since it satisfies $\mathcal{A}$ ($\mathcal{A} \sqsubseteq \mathcal{A}$), completeness follows.

The abstraction relation in the framework based on simulation is still language membership: a Kripke Structure $\mathcal{S}$ is abstracted by automaton $\mathcal{A}$ iff $\mathcal{S} \in \mathcal{L}(\mathcal{A})$. However, this can be replaced by an equivalent definition in terms of simulation. For this, we need to be able to "lift" a Kripke Structure to an automaton. The structure states become BRANCH states of the automaton, and trivial OR states are inserted that each have only a single OR choice.

Definition 5. *Let $\mathcal{S} = (S, \hat{S}, R, L)$ be a Kripke Structure. The* automaton associated with $\mathcal{S}$, *$\mathcal{A}ut(\mathcal{S})$, is as follows. $\mathcal{A}ut(\mathcal{S})$ has* OR *states $\{q_s \mid s \in S\}$,* BRANCH *states $\{b_s \mid s \in S\}$, and initial states $\{q_{\hat{s}} \mid \hat{s} \in \hat{S}\}$. Each* OR *state q_s has b_s as its only* OR *choice. Each* BRANCH *state b_s has a* BR *transition to q_t for every t such that $R(s,t)$ in the Kripke Structure. The labeling of a* BRANCH *state b_s is the labeling of s in the Kripke Structure. The indexing function assigns 0 to every* OR *state.* □

It can be shown that $\mathcal{A}ut(\mathcal{S})$ accepts precisely the bisimulation class of $\mathcal{S}$. We now have:

Lemma 1. *$\mathcal{S} \in \mathcal{L}(\mathcal{A})$ iff $\mathcal{A}ut(\mathcal{S}) \sqsubseteq \mathcal{A}$.*

Proof. (sketch) The simulation game for $\mathcal{A}ut(\mathcal{S}) \sqsubseteq \mathcal{A}$ is identical to an automaton run in this special case where the automaton on the left hand side is obtained from a Kripke Structure.

4 Translations: KMTS's to Automata

In the previous section, we gave a simple translation from Kripke Structures to automata. In this section we present a more elaborate translation from *Kripke Modal Transition Systems (KMTS's)* [17] to automata. This provides insight into their relation, and can be adapted to obtain translations from similar notions, such as the Disjunctive Modal Transition Systems [23]. KMTS's are based on 3-valued logic. Let $\mathbf{3} = \{true, maybe, false\}$, and define the *information ordering* $\leq$ by: $maybe \leq x$, $x \leq x$ for every $x \in \mathbf{3}$, and $x \not\leq y$ otherwise. $\leq$ is lifted in the standard way to functions into $\mathbf{3}$, and $\geq$ denotes the inverse of $\leq$. A proposition p in a state of a KMTS takes on values in $\mathbf{3}$. We formalize this by letting p be a 3-valued predicate that maps states to $\mathbf{3}$. The following definitions are adapted from [13].

Definition 6 ([13]). *A* Kripke Modal Transition System *is a tuple $M = (S, \hat{S}, \longrightarrow, \dashrightarrow, P)$, where S is a nonempty countable set of* states, *$\hat{S} \subseteq S$ is a subset of* initial states, *$\longrightarrow, \dashrightarrow \subseteq S \times S$ are the* must *and* may *transition relations resp., such that $\longrightarrow \subseteq \dashrightarrow$, and $P = \{pred_p \mid p \in Prop\}$, where for every $p \in Prop$, $pred_p : S \to \mathbf{3}$.* □

With $\longrightarrow = \dashrightarrow$ and P the 2-valued predicates $pred_p : S \to \{true, false\}$ we recover the Kripke Structures from Definition 1.

We usually just write p for $pred_p$. Finite state KMTS's have been suggested as abstract objects in a framework where the concrete objects of interest are Kripke Structures and the properties are phrased in branching-time temporal logic. The concretization of a KMTS M (the set of Kripke Structures that are abstracted by M) is called its *completion*, defined as follows.

Definition 7 ([13]). *The* completeness preorder $\succeq$ *between states of KMTS's* $M_1 = (S_1, \hat{S}_1, \longrightarrow_1, \dashrightarrow_1, P_1)$ *and* $M_2 = (S_2, \hat{S}_2, \longrightarrow_2, \dashrightarrow_2, P_2)$ *is the greatest relation* $B \subseteq S_1 \times S_2$ *such that* $(s_1, s_2) \in B$ *implies: (1) for every* $p \in Prop$, $p(s_1) \geq p(s_2)$*; (2)* $\longrightarrow_1$ *simulates* $\longrightarrow_2$*; and (3)* $\dashrightarrow_1$ *is simulated by* $\dashrightarrow_2$*.* M_1 *is* more complete *than* M_2*, denoted* $M_1 \succeq M_2$*, iff for every* $\hat{s}_1 \in \hat{S}_1$ *there exists* $\hat{s}_2 \in \hat{S}_2$ *with* $\hat{s}_1 \succeq \hat{s}_2$*. The* completion $C(M)$ of M *is the set of all Kripke Structures* $\mathcal{S}$ *such that* $\mathcal{S} \succeq M$*.* □

The final component we need for a KMTS-based abstraction framework is an interpretation of branching-time properties over a KMTS. Also in this case, the setting is 3-valued. We consider here the so-called *thorough* semantics since it has a more direct connection to automata.

Definition 8 ([13]). *The* thorough semantics *assigns a truth value from* **3** *to a KMTS and a temporal formula, as follows:* $[M \models \varphi] = true$ *(false) if* $\mathcal{S} \models \varphi$ *is true (resp. false) for all* $\mathcal{S} \in C(M)$*, and* $[M \models \varphi] = maybe$ *otherwise*[6]*.* □

Note that the relations $\longrightarrow$ and $\dashrightarrow$ can together be seen as a 2-bit encoding of a single "3-valued transition relation", i.e. a predicate that maps every pair s, t of states into **3**: when $s \longrightarrow t$ and $s \dashrightarrow t$, this transition relation has value *true*; when neither $s \longrightarrow t$ nor $s \dashrightarrow t$, it has value *false*; and when only $s \dashrightarrow t$, it is *maybe*; note that the fourth combination is excluded by the requirement that $\longrightarrow \subseteq \dashrightarrow$. In terms of this 3-valued transition relation, the intuition behind a KMTS's transition structure can be explained as follows. View a state s of a KMTS as the set of all Kripke Structure states that it abstracts (i.e., $\{n \mid n \succeq s\}$). Consider the value of the transition between KMTS states s and t. If it is *true*, then all states in s have a transition to some state in t. If it is *false*, then *no* state in s has a transition to any state in t. If it is *maybe*, then some states in s do, and others do not, have a transition to some state in t.

We can "massage" a KMTS into an automaton by splitting its states so that all propositions and transitions become definite (*true* or *false*). What makes this possible[7] is the presence of OR states: Initially, trivial OR states (with single

[6] Compared to the definitions in [17] and [13], we have added initial states to KMTS's. In this case, a temporal property is defined to be *true* (*false*) for a KMTS if it is *true* (resp. *false*) in all initial states, and *maybe* otherwise.

[7] This is not possible in general if we stay within the framework of KMTS's, which can be seen by considering the KMTS that has a single state where the (single) proposition p is *maybe*, and which has *must* and *may* transitions back to itself. There exists no *finite* 2-valued KMTS that has the same completion — the set of all total trees labeled with subsets of $\{p\}$.

successors) are inserted into the KMTS — one OR state preceding every KMTS state — similar to the definition of the transformation $\mathcal{A}ut$ of Definition 5. Consider a state s and a proposition p that evaluates to *maybe* in s, see Figure 2(a). If we view s as its concretization some of its concrete states will evaluate p to

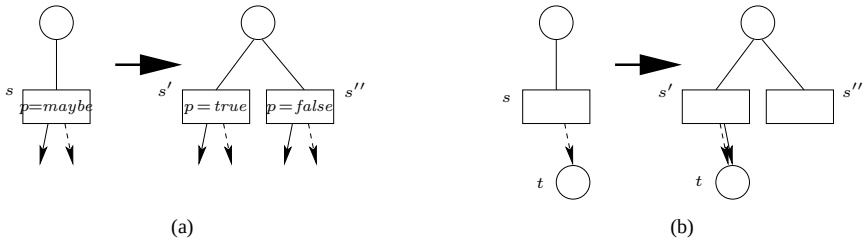

Fig. 2. Removing 3-valuedness from propositions (a) and transitions (b) by splitting states. Must and may transitions are depicted as solid and dashed arrows resp.

true, and others to *false*. We split s so as to separate these states from one another, creating new abstract states s' and s'' in place of s, one where p is *true*, and another where it is *false*. All transitions from s are copied to both s' and s''. Similarly, a state s with an outgoing *maybe* transition (to OR state t, say) is replaced by two states, one with a *true* transition to t, the other with a *false* transition to t; see Figure 2(b).

This is done for every state s, every proposition that is *maybe* in s, and every outgoing *maybe* transition from s. This translation, called τ, will be defined more formally below. While it turns the completion of a KMTS into the language of the resulting automaton, i.e. $C(M) = \mathcal{L}(\tau(M))$, it does not follow that M and $\tau(M)$ make the same properties true. For this, we need to generalize the semantics $\models^\alpha$ of branching-time properties, interpreted over automata, to be 3-valued as well.

Definition 9. *The 3-valued semantics $\models^\alpha$ maps an automaton and a branching-time property φ (expressed by an automaton $\mathcal{A}_\varphi$) to* **3***, as follows.* $[\mathcal{A} \models^\alpha \mathcal{A}_\varphi] =$ *true (false) if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$ (resp. $\mathcal{L}(\mathcal{A}) \subseteq \overline{\mathcal{L}(\mathcal{A}_\varphi)}$) and maybe otherwise.* □

Theorem 5. *For every KMTS M and branching-time temporal property φ,* $[M \models \varphi] = [\tau(M) \models^\alpha \mathcal{A}_\varphi]$.

4.1 Modal Automata

The description of τ above can be seen as a two-step process. First, by inserting OR states, the KMTS is lifted into a special kind of automaton, namely one that allows 3-valued propositions and transitions. Then, this 3-valuedness is compiled away by splitting BRANCH states.

Definition 10. *A* modal automaton $\mathcal{A}$ *is a tuple* $(Q, B, \hat{Q}, \text{OR}, \longrightarrow, \dashrightarrow, P, \Omega)$ *where Q, B, $\hat{Q}$,* OR*, Ω are as in Definition 2, and:*

– $\longrightarrow, \dashrightarrow \subseteq B \times Q$ *are* must *and* may *transition relations, resp., from* BRANCH *states to* OR *states, and*
– $P = \{pred_p \mid p \in Prop\}$*, where for every* $p \in Prop$*,* $pred_p : B \to \mathbf{3}$. □

The notion of automaton simulation from Definition 4, $\sqsubseteq$, is extended to these modal automata.

Definition 11. *The simulation relation* $\sqsubseteq$ *on modal automata is defined as in Definition 4 where* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are now modal automata, and rule 3 is replaced by the following:*

*3. (*BRANCH*) In a "*BRANCH*" configuration* (b_1, b_2) *(where* $b_1 \in B_1$ *and* $b_2 \in B_2$*), each of the following are continuations of the play:*
   (a) (Prop) In this continuation, the play ends and is a win for Player I if for all $p \in Prop$*,* $p(b_1) \geq p(b_2)$*, and it is a win for for Player II otherwise.*
   (b) (may) Player II chooses an OR *state* q_1 *such that* $b_1 \dashrightarrow_1 q_1$*; Player I must respond with an* OR *state* q_2 *such that* $b_2 \dashrightarrow_2 q_2$*, and the play continues from configuration* (q_1, q_2)*.*
   (c) (must) Player II chooses an OR *state* q_2 *such that* $b_2 \longrightarrow_2 q_2$*; Player I must respond with an* OR *state* q_1 *such that* $b_1 \longrightarrow_1 q_1$*, and the play continues from configuration* (q_1, q_2)*.*

The language $\pounds(\mathcal{A})$ *of a modal automaton* $\mathcal{A}$ *is the set of all Kripke Structures* $\mathcal{S}$ *such that* $\mathcal{A}ut(\mathcal{S}) \sqsubseteq \mathcal{A}$. □

τ is now defined as the composition of translations τ_1 and τ_2:

τ: KMTS $\xrightarrow{\tau_1}$ modal (i.e., 3-valued) automaton $\xrightarrow{\tau_2}$ (2-valued) automaton

The definition of τ_1 is a generalization of the embedding $\mathcal{A}ut$ of Def. 5.

Definition 12. *Let* $M = (S, \hat{S}, \longrightarrow, \dashrightarrow, P)$ *be a KMTS. The* modal automaton associated with M*,* $\tau_1(M)$*, is as follows:* $\tau_1(M)$ *has* OR *states* $\{q_s | s \in S\}$*,* BRANCH *states* $\{b_s \mid s \in S\}$*, and initial states* $\{q_{\hat{s}} \mid \hat{s} \in \hat{S}\}$*. Each* OR *state* q_s *has* b_s *as its only* OR *choice. Each* BRANCH *state* b_s *has a* $\longrightarrow$ *(*$\dashrightarrow$*) transition to* q_t *for every* t *such that* $\longrightarrow (s, t)$ *(resp.* $\dashrightarrow (s, t)$*) in the KMTS. The predicates of* $\tau_1(M)$ *are the same as in the KMTS. The indexing function assigns* 0 *to every* OR *state.* □

Lemma 2. *Let* M *be a KMTS. Then* $C(M) = \pounds(\tau_1(M))$.

Proof. Similar to the proof of Lemma 1.

The translation τ_2, that compiles away 3-valued propositions and transitions from a modal automaton while preserving its language, is itself defined in two steps. The first step, τ_{2A}, removes 3-valuedness from the propositional labeling, yielding modal automata that have *must* and *may* transitions, but whose state predicates assign definite values to states.

Definition 13. *Let* $\mathcal{A} = (Q, B, \hat{Q}, \text{OR}, \longrightarrow, \dashrightarrow, P, \Omega)$ *be a modal automaton. For* $b \in B$*, define its associated* valuation $val(b) : Prop \to \mathbf{3}$ *to be* $\lambda\ p \in Prop\ .\ pred_p(b)$*. For a valuation* $v : Prop \to \mathbf{3}$*, define its* completion $Compl(v)$ *as* $\{w : Prop \to \mathbf{2} \mid w \geq v\}$*. With every state* $b \in B$ *we associate a fresh set* $C(b) = \{b_w \mid w \in Compl(val(b))\}$ *of states. The modal automaton* $\tau_{2A}(\mathcal{A})$ *is defined as follows.*

- *its set of* OR *states is* Q*,*
- *its set of* BRANCH *states is* $\bigcup_{b \in B} C(b)$*,*
- *its set of initial states is* $\hat{Q}$*,*
- *its choice relation is* $\{(q, b_w) \mid (q, b) \in \text{OR}, b_w \in C(b)\}$*,*
- *its* must *transition relation is* $\{(b_w, q) \mid b \longrightarrow q, b_w \in C(b)\}$*,*
- *its* may *transition relation is* $\{(b_w, q) \mid b \dashrightarrow q, b_w \in C(b)\}$*,*
- *for every* $p \in Prop$ *and* $b_w \in$ BRANCH*,* $pred_p(b_w) = w(p)$*, and*
- *its indexing function is* $\lambda\ q\ .\ 0$*.* □

Lemma 3. *Let* $\mathcal{A}$ *be a modal automaton. Then* $\pounds(\mathcal{A}) = \pounds(\tau_{2A}(\mathcal{A}))$*.*

The kind of automata targeted by τ_{2A} have actually been defined in [19], where it is also shown that they can be translated back to (non-modal) automata. We reuse their result here, to emphasize the close relationship between the use of 3-valuedness for abstraction in verification, and developments in the field of automata theory. In [19], the BRANCH transitions from a state t which has OR successors $q_1, \ldots, q_k$ are given as a first-order logic formula of the form $\exists n_1, \ldots, n_k.\ q_1(n_1) \wedge \cdots \wedge q_k(n_k)\ \wedge\ \forall z.\ \beta(z)$ which specifies when a tree node x is accepted from the automaton state t. The variables n_i and z range over the child nodes of x. The notation $q(n)$ means that (the tree rooted at) child n is accepted from state q, and $\beta(z)$ is a conjunction of disjunctions of formulas of the form $q(n)$. Thus, the formula says that for every $1 \leq i \leq k$, there exists a child n_i of x that is accepted from q_i, and in addition the constraint β must hold for all children of x. Note that the automata from Def. 2 are a special case where the β is of the form $\exists 1 \leq i \leq k.\ q_i(z)$. Furthermore, *must* and *may* transitions can be recovered as well. Let $Q_{must} = \{q'_1, \ldots, q'_l\}$ be a subset of $\{q_1, \ldots, q_k\}$. The formula $\exists n_1, \ldots, n_l.\ q'_1(n_1) \wedge \cdots \wedge q'_l(n_l) \wedge \forall z\ \exists 1 \leq i \leq k.\ q_i(z)$ then specifies that the states in Q_{must} act as *must* successors, while all q_i are *may* successors of t. Hence this special form of automaton is the same as a modal automaton with *must* and *may* transitions, but in which all propositional valuations are definite. In [19] it is shown that such automata can be translated back into the restricted form of Def. 2, i.e., calling the translation τ_{2B}, it is shown that $\pounds(\mathcal{A}) = \pounds(\tau_{2B}(\mathcal{A}))$. Together with Lemma 3, this implies the correctness of τ_2 which is defined as τ_{2A} followed by τ_{2B}.

Lemma 4. *Let* $\mathcal{A}$ *be a modal automaton. Then* $\pounds(\mathcal{A}) = \pounds(\tau_2(\mathcal{A}))$*.*

Finally, Theorem 5 follows easily from Lemma 2 and Lemma 4.

5 Maximal Model Theorems

In the previous sections we have proposed to use automata as the abstract objects in an abstraction framework for checking branching-time properties over transition systems. In the resulting framework, the concrete and abstract objects do not live in strictly separated worlds: through the embedding $\mathcal{A}ut$ from Definition 5, transition systems themselves are promoted to automata — be it possibly infinite ones. It follows from Lemma 1 that the transition system and its embedding satisfy the same branching time properties, regardless of whether these are evaluated using the language inclusion or the simulation based definition. Furthermore, Lemma 1 and Theorem 2 together show that the abstraction relation between the concrete and abstract domains can be embedded into the *abstraction order* $\sqsubseteq$ over the abstract domain.

Another area that can benefit from this view is the study of *maximal models*. There, we also consider objects ordered by an abstraction order which is such that more-abstract objects satisfy fewer properties, relative to a given logic of interest. A *maximal model* for a property is then a model for the property that is maximally abstract. In abstraction frameworks such as the ones above, where the concrete domain is embedded in the abstract domain, there is a close connection between maximal models and completeness, made explicit below.

Theorem 6. *If every property has a finite maximal model, then the abstraction framework is complete.*

Proof. Let M be a model of property ϕ, and let max_ϕ be a finite maximal model for ϕ. By definition of maximality, max_ϕ abstracts M. Thus, one can always pick $\max_\phi$ as a finite abstraction for M, ensuring completeness.

The converse is not necessarily true.Grumberg and Long showed [15] how to construct finite, fair Kripke Structures that are maximal models for ACTL (the universal fragment of CTL) through tableaux constructions; this was extended to ACTL* by Kupferman and Vardi [21], using a combination of tableaux and automata-theoretic constructions. By the theorem above, it follows immediately that fair simulation abstraction yields a complete framework for ACTL, ACTL*, and linear-time temporal logic (which can be considered to be a sublogic of ACTL*). For the richer branching time logics that include existential path quantification, however, there cannot be such maximal models.

Theorem 7. *In abstraction frameworks for branching time logics that are based on Kripke Structures or Kripke Modal Transition Systems, not every property has a finite maximal model.*

Proof. This follows immediately from Theorem 6 and the result of [6] showing that these frameworks are incomplete for existential properties.

One can recover the guarantee of finite maximal models, however, by enlarging the class of structures to include automata[8]. A Kripke Structure M is now viewed as the automaton $\mathcal{A}ut(M)$.

Theorem 8. *In the automaton abstraction frameworks based either on language inclusion or on simulation, every property has a finite maximal model.*

Proof. Consider a property given as a finite automaton $\mathcal{B}$. Viewed as a structure, $\mathcal{B}$ satisfies the property $\mathcal{B}$ in either framework. For any other model M of $\mathcal{B}$, letting the satisfaction and simulation relations coincide for automata models, $\mathcal{B}$ is maximal in the abstraction order. Hence, $\mathcal{B}$ is a finite maximal model for property $\mathcal{B}$ in either framework.

We can use the connection made in Theorem 8 to re-derive the maximal model results for ACTL and ACTL*; in fact, we can extend these easily to the universal fragment of the mu-calculus, A_μ. It should be noted that A_μ is more expressive than even ACTL* (e.g., "every even-depth successor satisfies P" $(\nu Z : p \wedge \mathsf{AX}(\mathsf{AX}(Z)))$ is expressible in A_μ but not in ACTL*). The idea is to (i) construct an equivalent finite automaton for a formula in this logic — this is using known techniques from automata theory —, (ii) view this automaton as a maximal model, following the theorem above, and then (iii) show that, due to its special structure, it can be transformed back to a Kripke Structure. The result is the following theorem. Its proof is omitted due to space constraints.

Theorem 9. *The fair simulation abstraction framework on Kripke Structures is complete for A_μ.*

6 Discussion

We have proposed to use tree automata as abstractions of countable transition systems, in the verification of branching time properties. A tree automaton serves as an abstraction for any transition system in its language. Expressing also branching time properties as tree automata, the definition of when a property holds on an abstraction can be defined as language inclusion, or alternatively as simulation between automata. Both frameworks are trivially sound. The notion of simulation between automata on infinite trees is novel to the best of our knowledge. Like in the word case, it is easier to decide than language inclusion, and is a sufficient condition for it.

Also completeness follows directly in both frameworks. The completeness argument shows that, for a transition system $\mathcal{S}$ and a property $\mathcal{A}_\varphi$ that is true of it, the finite abstraction of $\mathcal{S}$ that can be used to demonstrate $\mathcal{A}_\varphi$ is $\mathcal{A}_\varphi$ itself. This highlights the essence of the more elaborate completeness arguments presented in [32, 20, 25, 6]. The use of Janin and Walukiewicz' μ-automata, whose

[8] This solution is similar to the introduction of complex numbers: enlarging the solution space from real to complex numbers ensures that every non-constant polynomial has a "zero".

languages are closed under bisimulation and therefore correspond naturally to the μ-calculus, further simplifies the technical presentation.

Section 4 demonstrated how Kripke Modal Transition Systems can be transformed into automata. Similar constructions can be carried out for the other transition notions, such as disjunctive modal transition systems. The insight gained from these transformations is that one can view the various proposals in the literature as being but variations on automaton syntax, some more compact than others. This point is implicit in our earlier paper [6], which demonstrates that *alternating* tree automata — the most compact automaton notation — correspond to Focused Transition Systems.

A key issue in the practice of verification via abstractions is how to automatically obtain suitable abstractions. By the embedding results of Section 4, any approach to the construction of abstractions (e.g. [30]) can be used in the automaton-based framework. While the problem is undecidable in general, the completeness result guarantees that a suitable automaton-as-abstraction always exists.

Going beyond the technical benefits of automata, we feel that viewing automata as abstract objects, and realizing that known notions are but automata in disguise, is a simple but profound shift of perspective that should enable many fruitful connections between abstraction and automata theory.

Acknowledgements We thank Nils Klarlund for his contributions to several discussion on the topic of this paper, and for his comments on an earlier draft. This work is supported in part by NSF grant CCR-0341658.

References

1. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *CAV*, volume 1633 of *LNCS*. Springer, 1999.
2. M. Chechik, S. Easterbrook, and V. Petrovykh. Model-Checking over Multi-valued Logics. In *FME*, volume 2021 of *LNCS*. Springer, 2001.
3. E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*. Springer-Verlag, 1981.
4. R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In *SAS*, volume 983 of *LNCS*. Springer, 1995.
5. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.
6. D. Dams and K.S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, 2004.
7. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19(2):253–291, 1997.
8. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In *J. of Logic and Algebraic Programming*, 52–53:109–127. Elsevier, 2002.
9. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, July 1996.

10. L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *LICS*, 2004.
11. E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *FOCS*, 1988. Full version in *SIAM Journal of Computing*, 29(1):132–158, 1999.
12. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, 1991.
13. P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *VMCAI*, volume 2575 of *LNCS*. Springer, 2003.
14. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *Journal of the ACM*, 47(2):361–416, 2000.
15. O. Grumberg and D.E. Long. Model checking and modular verification. In *ACM TOPLAS*, 1994.
16. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *CONCUR*, volume 1243 of *LNCS*. Springer, 1997.
17. M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, volume 2028 of *LNCS*. Springer, 2001.
18. D. Janin and I. Walukiewicz. Automata for the modal mu-calulus and related results. In *MFCS*, volume 969 of *LNCS*. Springer, 1995.
19. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR*, volume 1119 of *LNCS*. Springer, 1996.
20. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163(1):203-243. Elsevier, 2000.
21. O. Kupferman and M.Y. Vardi. Modular model checking. In *COMPOS*, volume 1536 of *LNCS*. Springer, 1997.
22. K.G. Larsen and B. Thomsen. A modal process logic. In *LICS*, 1988.
23. K.G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, 1990.
24. R. Milner. An algebraic definition of simulation between programs. In *2nd IJCAI*. William Kaufmann, 1971.
25. K.S. Namjoshi. Abstraction for branching time properties. In *CAV*, volume 2725 of *LNCS*. Springer, 2003.
26. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
27. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th Intl. Symp. on Programming*, volume 137 of *LNCS*. Springer-Verlag, 1982.
28. D. A. Schmidt. Closed and logical relations for over- and under-approximation of powersets. In *SAS*, volume 3148 of *LNCS*. Springer, 2004.
29. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19:424–437, 1990.
30. S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS*, volume 2988 of *LNCS*. Springer, 2004.
31. R.S. Streett and E.A. Emerson. The propositional mu-calculus is elementary. In *ICALP*, volume 172 of *LNCS*, 1984. Full version in *Information and Computation* 81(3): 249-264, 1989.
32. T.E. Uribe. *Abstraction-Based Deductive-Algorithmic Verification of Reactive Systems*. PhD thesis, Stanford University, 1999.