The Existence of Finite Abstractions for Branching Time Model Checking

Dennis Dams Kedar S. Namjoshi
Bell Labs, Lucent Technologies
Murray Hill, NJ, USA
{*dennis,kedar*}*@research.bell-labs.com*

Abstract

Abstraction is often essential to verify a program with model checking. Typically, a concrete source program with an infinite (or finite, but large) state space is reduced to a small, finite state, abstract program on which a correctness property can be checked. The fundamental question we investigate in this paper is whether such a reduction to finite state programs is always possible, for arbitrary branching time temporal properties.

We begin by showing that existing abstraction frameworks are inherently incomplete for verifying purely existential or mixed universal-existential properties. We then propose a new, complete abstraction framework which is based on a class of focused transition systems (FTS's). The key new feature in FTS's is a way of "focusing" an abstract state to a set of more precise abstract states. While focus operators have been defined for specific contexts, this result shows their fundamental usefulness for proving non-universal properties. The constructive completeness proof provides linear size maximal models for properties expressed in logics such as CTL and the mu-calculus. This substantially improves upon known (worst-case) exponential size constructions for their universal fragments.

1. Introduction

In this paper, we explore the role played by program abstraction in enabling verification through model checking. Model checking [6, 39] has the advantage of being able to decide complex temporal properties fully automatically for finite systems. There has been significant progress in the past two decades on enhancing the scalability of model checking for finite state programs, and on developing algorithms for classes of infinite state programs (such as timed and hybrid automata, parameterized processes, and pushdown systems); however, abstraction is still an essential part of the verification process. An *abstraction framework* is given by: a class of programs, an abstraction relation between programs, a class of properties, and a satisfaction relation defining which properties hold of a program. Such a framework is required to be *sound*, i.e., a property that holds of an abstract program should hold of more concrete ones.

A fundamental question in the context of model checking is whether a framework is *complete*: i.e., is it always possible to find a small, finite-state abstract program that is precise enough to prove a correctness property? For infinite concrete programs, the key question is whether a finite abstract program can be constructed at all; for finite programs, the key question is how small the abstract program can be, while yet being precise enough. We explore this question for the class of branching time properties, expressible in logics such as CTL and the propositional mu-calculus, and by finite-state automata on infinite trees.

In the first part of this paper we show, with simple examples, that several well-known abstraction frameworks are incomplete for proving branching time properties. These include frameworks that use *simulation* [34], *bisimulation* [37, 21], *modal (may-must)* [28], *partial (3-valued)* abstraction [4], or *mixed* notions [7, 9, 23]. The negative result holds for purely existential and mixed universal-existential formulas, for both safety and liveness properties, and even with the addition of fairness restrictions or stuttering insensitivity. For purely universal properties, it is known that fair simulation suffices for completeness, as is described in more detail in the related work section.

To achieve completeness, therefore, we propose an abstraction framework that is based on a new class of *focused* transition systems (FTS's, for short). As in a mixed transition system, an FTS has two transition relations, one representing an over-approximation of the concrete relation, the other an under-approximation; these are used to interpret universal and existential path quantifiers, respectively. The key new feature is a "focus" map, which relates an abstract state to a set of more precise abstract states, and a dual "de-focus" capability. Focus steps can be seen as splitting an abstract state into sub-cases, breaking up a proof obligation into simpler parts. De-focus steps may be seen as generalization (and thus strengthening) of an obligation, which can

also simplify the proof. These new capabilities require extensions of the known refinement and property satisfaction relations for mixed transition systems.

The focus and de-focus operations are not "rabbits" (as Edsger W. Dijkstra was fond of calling concepts that seem to be pulled out of a hat ...!) — they are the transition system analogues of the OR and AND operators in alternating tree automata. One may indeed (almost) identify property automata with abstract transition systems: an automaton *accepts* a set of transition systems, whereas an abstract transition system *abstracts* a set of transition systems.

The completeness proof relies on this connection. It shows how to turn an alternating tree-automaton property ϕ that holds of a transition system M, into a finite FTS $\overline{\phi}$ which abstracts M and satisfies ϕ. As $\overline{\phi}$ is independent of M, this construction produces, in fact, an FTS that is a *maximal model* of the property: $\overline{\phi}$ is maximal in the sense that it abstracts all other models of the property, and it is a model because it satisfies the property. This construction produces, through translations to alternating tree automata [15], maximal models that are of size linear in the formula size for formulas in CTL and the mu-calculus, and exponential in formula size for CTL* formulas. The maximal model constructions for the universal fragments of CTL [20] and CTL* [26] generate models through translations to non-deterministic tree automata. This results in maximal models that are ordinary transition systems and these are (worst-case) exponentially larger than our FTS maximal models.

These exponential size improvements suggest advantages to using FTS's over ordinary transition systems, even for abstracting universal properties. Indeed, focus operations have been defined for specific contexts, such as the static analysis of heap shapes [40]. For instance, an abstract state denoting a linked list shape about which nothing is known may be focused into a pair of abstract states, specifying whether the list is empty or not. The formulations in this paper provide a general basis for reasoning about such focus operators, and the structure of the completeness proofs may help suggest mechanizable heuristics for defining focus operators.

1.1. Related Work

We describe closely related work, including other completeness results, and the study of precision of abstractions. As mentioned earlier, existing maximal model theorems yield completeness for some universal branching-time logics. In [20], a maximal model theorem, yielding a worst-case exponential size model, is shown for the universal sublogic of CTL. This is extended in [26] through automata-theoretic constructions to a maximal model construction for the universal fragment of CTL* (which includes linear-time temporal logic, LTL [38]). As noted in [30], these constructions may be viewed as turning an edge-labeled, non-deterministic tree automaton for a property into a state-labeled transition system model simply by splitting each automaton state into several copies, one for each transition label, and by appropriately redirecting transitions. Our completeness proof follows a similar construction; however, the end result is a focused transition system, rather than an ordinary one — the incompleteness results rule out the existence of an ordinary, finite maximal model.

The formulation of abstraction-aided model checking given here may be called *model abstraction*, since it seeks to abstract a concrete model of a formula. It is not, however, the only possible way of combining the two concepts. Another approach, which may be called *game abstraction*, abstracts the AND-OR game graph that is obtained by the game formulation of model checking [17]. The abstraction relation needs to preserve the existence of a winning strategy, rather than all formulas of a logic. An important case is that of linear-time logic. Given a property ϕ and transition system M, the usual model checking strategy is to first form the product of M with the word automaton $\mathcal{A}_{\neg\phi}$ for the negation of ϕ, and check if this product has no accepting computations. The product can be viewed as a AND-only game graph (i.e., without OR moves), where a winning strategy shows that there are no accepting computations. In [24, 25, 43], it is shown how to construct a finite-state abstract program, with fairness constraints, which fair-simulates the product of a concrete program with a negated property automaton. The abstraction relation is such that, if the abstract program has no fair computations (i.e., a winning strategy), so does the original one.

This abstraction approach is generalized by the second author in [36] to parity games defined by the game-theoretic formulation of mu-calculus model checking. The approach is to over-approximate the AND moves and under-approximate the OR moves, thus preserving the existence of a winning strategy. A subtlety uncovered in this work is that the standard under-approximation definition[1] *does not work* in general. It is necessary to split abstract OR moves into cases, and consider each case when formulating a winning strategy. The focus operations in FTS's can be thought of as the model-based analogue of this OR splitting.

Completeness is closely related to the problem of determining the precision of abstractions. Informally stated, the problem is to define a precision pre-order on abstract systems, such that more precise systems satisfy more properties. Precision of modal and mixed transition systems is studied in [7, 9]. The connection is that, in studying completeness, one is trying to show that a precise enough finite abstract system always exists. The papers above show that

[1] I.e., for an abstract OR transition from state a to a', every concrete state s related to a has a concrete OR transition to a state s' related to a'.

precision may be improved either by adding more precise abstract states or by restricting transitions, but they do not resolve the completeness question. Indeed, our results show that completeness cannot be achieved for these abstraction frameworks, and the introduction of new features such as focus operators is necessary. In [29, 11, 41], operators similar to focus are added to abstract systems in order to improve precision, but also there completeness is not pursued; indeed, since these proposals for abstract systems do not feature a notion of fairness, they are not complete for liveness properties.

Readers aware of small-model theorems for tree automata [22, 12] may wonder why completeness does not follow trivially from these theorems, which show that a small finite model exists for any satisfiable property. The missing ingredient is abstraction: there may be no clear abstraction relationship between the small model, M', whose existence is guaranteed by these theorems, and the given concrete transition system M. For instance, a 1-state system satisfying proposition Q is a small model for the CTL property $\mathsf{EF}(Q)$ ("there is a reachable Q-state"), but by no known abstraction notion is it related to all concrete models where Q-states are reachable only after one or more steps.

2. Incompleteness Results

In this section we consider a general notion of abstraction to demonstrate that many commonly used abstraction frameworks are incomplete for branching-time logics. The concrete semantics of a program is expressed by a (possibly infinite) transition system $M = (S, I, \rightarrow, L)$ where L labels each state $s \in S$ by the set of atomic propositions (from a given finite set AP) that hold in s. I is the nonempty set of initial states. On the abstract side, we consider *mixed transition systems* $A = (S_A, I_A, \xrightarrow{must}, \xrightarrow{may}, L_A^-, L_A^+)$, whose states, in S_A, are called *abstract states*. A is called finite iff S_A is finite. A *mixed simulation relation* from a concrete system M to a mixed transition system A, as above, is any relation $\rho \subseteq S \times S_A$ for which each of the following requirements holds: (i) every concrete initial state (in I) is ρ-related to some abstract initial state (in I_A); (ii) whenever $\rho(c, a)$ then $L_A^-(a) \subseteq L(c) \subseteq L_A^+$ — intuitively, the propositions in $L_A^-(a)$ are those known to be true and those in $AP \setminus L_A^+(a)$ are known to be false; (iii) if $\rho(c, a)$ and $a \xrightarrow{must} a'$, there exists c' s.t. $c \rightarrow c'$ and $\rho(c', a')$, so the *must* relation under-approximates the concrete transition relation; and (iv) if $\rho(c, a)$ and $c \rightarrow c'$, there exists a' s.t. $a \xrightarrow{may} a'$ and $\rho(c', a')$, so the *may* relation is an over-approximation. A relation ρ is a *simulation* if (i), (ii), and (iv) hold, and a *reverse simulation* if (i), (ii), and (iii) hold. These relations between states are lifted to the following three abstraction relations between concrete and mixed transition systems M and A as above. A *mix-simulates* M if there exists a mixed simulation relation from S to S_A. If there exists a simulation from S to S_A and $\xrightarrow{must}$ is empty, then A *simulates* M. If there exists a reverse simulation from S to S_A and $\xrightarrow{may}$ is empty, then A *reverse-simulates* M.

We consider branching-time temporal logics such as CTL, CTL*, and their existential and universal fragments (denoted ECTL, ACTL, etc.), which are interpreted over concrete transition systems as usual (cf. [13]); in particular, a formula is true of a system iff it holds in all of its initial states. To interpret a formula φ over a mixed transition system A as above, φ is first converted to positive normal form by the usual rewrite rules. The rules for satisfaction of atomic propositions and their negations, and of the temporal path quantifications differ from the usual definition. For $a \in S_A$ and $P \in AP$, $a \models P$ iff $P \in L_A^-(a)$, and $a \models \neg P$ iff $P \notin L_A^+(a)$. Existential quantifications are interpreted over *must* paths, and universal quantifications over *may* paths. Note that this results in a 3-valued notion of truth where $A \not\models \varphi$ is not equivalent to $A \models \neg\varphi$.

Let TL be a temporal logic, or a class of temporal formulas. An abstraction relation is *complete for TL* if for every concrete transition system M and every $\varphi \in$ TL such that $M \models \varphi$, there exists a *finite* A that abstracts M according to the relation, such that $A \models \varphi$.

Starting with [28], transition systems with dual transition relations have been used in the definition of various behavioral refinement relations that are sensitive to branching. Examples are (and this list is not intended to be complete): [7, 10, 19, 23]. There are some elements of these definitions that are not covered by our mixed transition system formulation. For example, [28, 19] consider *edge-labeled* systems; and many of the mentioned papers define the behavioral refinement relation between two mixed transition systems. However, these differences are either irrelevant to the incompleteness results, or the incompleteness results can be easily extended to those cases. Thus, we claim that the results presented below imply incompleteness of each of the varieties of mixed transition systems and the corresponding refinement notions. We do not consider here abstraction notions and logics for alternating transition systems [2, 1], which have a different flavor, in that they describe properties of the interaction of an open program with its environment.

For definitions of the notions of *safety* and *liveness* in the branching-time setting, see e.g. [33].

Theorem 1 *Abstraction through reverse simulation is incomplete for ECTL safety properties.*

Proof Consider a transition system M with the following structure. There are infinitely many initial states, numbered $(0, 0), (0, 1), \ldots$. For every n, there is a single path: $(0, n) \rightarrow (1, n) \rightarrow \ldots \rightarrow (n + 1, n)$ of $n + 2$ states, which

ends in a self-loop on $(n+1, n)$. The last state, $(n+1, n)$, satisfies only proposition Q; all others satisfy only proposition P. Then M satisfies the ECTL safety property ϕ: $\mathsf{E}(P\mathsf{W}Q)$, which asserts that from every initial state there is a path along which P holds so long as Q does not (W is the *weak Until*).

Assume that there is a *finite-state* mixed transition system $\overline{M}$ such that $\overline{M}$ reverse-simulates M and $\overline{M} \models \phi$; we will derive a contradiction from this. Let ρ denote the underlying mixed simulation relation from M's to $\overline{M}$'s states. Since $\overline{M}$ is finite and M has infinitely many states, there exists (by requirement (i) in the definition of reverse simulation) an initial state a_0 of $\overline{M}$ which is ρ-related to infinitely many of M's initial states; denote the set of these states by $\gamma(a_0)$. By the assumption that $\overline{M} \models \phi$, there exists a *must*-path $a_0, a_1, \ldots$ which is either infinite and $a_i \models P$ for every $i \geq 0$ (case A), or there exist a number $K \geq 0$ such that $a_K \models Q$ (case B). Because $\overline{M}$ reverse-simulates M, from every state c_0 in $\gamma(a_0)$ there must be a path $c_0, c_1, \ldots$ such that $\rho(c_i, a_i)$ for every i. By requirement (ii), propositional labelings are more precise in M, so in case A there must be an infinite path from every $c_0 \in \gamma(a_0)$ on which every state satisfies P, which contradicts the definition of M; and in case B a Q state is reached in at most K steps from every $c_0 \in \gamma(a_0)$, which is also a contradiction. □

The system M used in this proof has an infinite number of initial states, but this is not essential. By slightly extending it and choosing a more complicated formula, a proof can be given in which the system has a single initial state (and no infinite branching). Namely, transform M into M' by adding the proposition R to only the initial states, chaining together the initial states: $(0,0) \rightarrow (0,1) \ldots$, and making $(0,0)$ the unique initial state of M'. Let ϕ' be the ECTL safety property $\mathsf{EG}(R \wedge \mathsf{EXE}((P \wedge \neg R)\mathsf{W}(Q \wedge \neg R)))$, which says that there is a path where R holds globally (G), and from each point on the path, there is an offshoot satisfying P unless ("weak-until") Q. Then $M' \models \phi'$, and it can be shown in a similar way as in the proof above that there exists no finite mixed transition system that reverse-simulates M' and satisfies ϕ'.

Also *stuttering simulations* [32, 3, 18], which allow a transition on one side to be matched by a sequence of transitions on the other, cannot be used to establish a complete abstraction framework. A technical problem is that these simulations do not preserve the *Next* operator of temporal logic. Moreover, we can alter the transition system and property from the proof of Theorem 1 to demonstrate incompleteness, as follows (without getting into the details of defining stuttering-insensitive abstractions). The path from each initial state of M is altered so that there is an alternation between P and R states before the final Q state is reached, and these alternations grow without bound for successive initial states. A property that holds for this system, which can be expressed in CTL*, is "*there exists a path along which segments of P states alternate with segments of R states, either forever, or until a Q state is reached*". Suppose that this holds for some finite "stuttering" abstraction of M. If the witness path reaches a Q state, it must have a bounded number of alternations of P and R; thus, each concrete path must have the same bound, a contradiction. If the segments alternate forever on the witness path, by finiteness, there must be a loop in the abstract system, which implies the existence of an infinite path in the concrete system, a contradiction as well.

2.1. Fair Abstraction Relations

Note that incompleteness for the sub-logic ECTL implies incompleteness for full CTL. Theorem 1 pinpoints existential safety formulas as one source. We show that another source of incompleteness are the liveness formulas in CTL, even when a notion of fairness is added. Intuitively, an abstraction reduces a system by "collapsing" multiple concrete states into a single abstract state. This may introduce spurious loops that prevent the demonstration of liveness properties. These loops can be excluded from consideration by fairness constraints. Adding fairness gives rise to complete frameworks for *universal* properties, as shown for linear time logics in [24, 25], and for ACTL and ACTL* in [20, 26], respectively.

We adapt the notion of mixed simulation to systems with fairness conditions. Call an infinite path in a concrete or mixed transition system *fair* if it satisfies the corresponding fairness condition. *Fair mixed simulation* is obtained by replacing the conditions (iii) and (iv) in the definition of mixed simulation by the following: (iii') if $\rho(c, a)$ and there exists a fair *must*-path $a_0, a_1, \ldots$ with $a_0 = a$, then there exists a fair path $c_0, c_1, \ldots$ with $c_0 = c$ such that $\rho(c_i, a_i)$ for every $i \geq 0$; and (iv') if $\rho(c, a)$ and there exists a fair path $c_0, c_1, \ldots$ with $c_0 = c$, then there exists a fair *may*-path $a_0, a_1, \ldots$ with $a_0 = a$ such that $\rho(c_i, a_i)$ for every $i \geq 0$. Fair versions of simulation and reverse simulation are derived from this as before. If systems are *machine closed*, meaning that every finite path can be extended to a fair path, then fair mixed simulation implies mixed simulation. The interpretation of temporal logic is adapted to systems with fairness conditions by relativizing the path quantifiers to the fair paths, as usual [16]. Since fairness constraints usually do not restrict finite computations (technically, this requires machine-closure), the previous incompleteness result continues to hold under fairness.

Theorem 2 *Abstraction through fair reverse simulation is incomplete for ECTL safety properties.*

Although fairness ensures completeness for universal properties, it does not help for existential ones.

Theorem 3 *Abstraction through fair reverse simulation is incomplete for ECTL liveness properties.*

Proof Consider the transition system M generated by the following program, with a single integer variable x: initially, x has an arbitrary value; in an infinite loop, the two nondeterministic actions of the program either increment or decrement x (by 1). All computationss are fair. The ECTL liveness property $\psi : \mathsf{EF}(x \geq 0)$ ("there is a fair path to a state where $x \geq 0$") is clearly true at every initial state of M. Let $\overline{M}$ be any fair, *finite-state* abstract program that fair reverse-simulates M. We claim that $\overline{M}$ cannot satisfy ψ, interpreted relative to fair paths. If it does, there is a fair path from some initial state of $\overline{M}$ that includes a state satisfying $(x \geq 0)$. But then this state must be reachable in a number of steps bounded by the size of $\overline{M}$. By reverse-simulation, the same bound applies to each initial state of M; a contradiction. □

Since the interpretation of existential properties does not depend on *may*-transitions, the more general notion of mixed simulation also falls short for both safety and liveness properties, even with the addition of fairness.

Theorem 4 *Abstraction through (fair) mixed simulation is incomplete for ECTL safety and liveness properties.*

So, the conclusion is that these incompleteness results are rather robust: Mixed simulation, in various forms, is not enough to provide a finite abstraction for every possible concrete system and existential property.

3. Completeness

In this section, we motivate and introduce the abstraction framework defined by the new class of focused transition systems, and show completeness for this framework.

The negative results in the previous section indicate that some new ingredient is needed for completeness. This ingredient comes from the observation that abstractions can (nearly) be identified with (tree) automata: an automaton *accepts* a set of transition systems, whereas an abstract system *abstracts* a set of transition systems. We define a class of "focused" transition systems that correspond closely to alternating tree automata. The crucial point to note is that for every mu-calculus property, there is a finite-state (in fact, linear-size) alternating tree automaton. Thus, a class of transition systems that closely corresponds to such automata is likely to satisfy the desired completeness property. (It is tempting to speculate that the converse is true as well: any abstraction framework that is complete is likely to correspond closely to the automaton framework – we do not formalize and prove this conjecture.)

To motivate and explain the formal constructions that follow, we show how to treat the incompleteness example

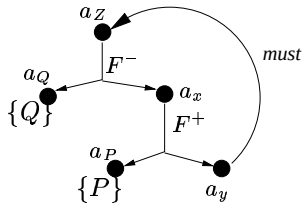

Figure 1. Example FTS

from Theorem 1. Let $\phi : \mathsf{E}(P\mathsf{W}Q)$ and M be the property and transition system defined in the proof of that theorem. This property can be expressed in the mu-calculus as $(\nu Z : Q \vee (P \wedge \mathsf{EX}(Z)))$, where $(\nu Z : \xi(Z))$ represents the greatest fixpoint of the function $\lambda Z.\xi(Z)$. Following the construction in [14], the derived alternating automaton has one state for each subformula, and a trivial acceptance condition. The initial state is q_Z, and the transition relation δ maps each state (i.e., subformula) to its defining expression (re-written in terms of automaton states): $\delta(q_Z) = q_Q \vee q_x$, $\delta(q_Q) = Q$, $\delta(q_x) = q_P \wedge q_y$, $\delta(q_P) = P$, and $\delta(q_y) = \mathsf{EX}\,(q_Z)$.

The justification for why M satisfies the automaton follows δ: a state satisfies q_Z if it either satisfies q_Q or q_x, and so forth; this can be formalized as an infinite game, as is done in the following section. Notice that the automaton state q_Z represents *all* program states that satisfy the property; these are not stratified according to their distance to a Q-state, because the automaton can dynamically "choose" at a state whether to classify it as one that immediately satisfies the property, placing it in the set of states corresponding to q_Q, or as one that can satisfy the property after at least one step (the states that correspond to q_x). This case-splitting flexibility results in a finite automaton description. The existing types of transition systems lack precisely this flexibility, forcing any abstraction to encode the stratification according to distance to Q, and thus be infinite. In focused transition systems, this flexibility is encoded by a *focus* step. The dual property, enjoyed by the $\wedge$ move in the automaton at q_x, is that of generalization. This represents alternation, which enables the linear-size (vs. exponential-size) representation of mu-calculus formulas. The generalization ability is represented in focused transition systems by *de-focus* steps, and should be similarly helpful in reducing the size of abstract systems. The other key property of automata, the presence of a global acceptance condition to check satisfaction of eventualities, corresponds nicely to a *fairness constraint* on focused transition systems. The other automaton operators correspond to the ordinary transitions in a focused transition system: an EX step in the automaton is similar to a *must* transition, while an AX step is similar to a *may* transition. (Note that, with this correspondence, not

every must-transition need coincide with a may-transition, as is required in modal transition systems.) From the automaton for ϕ, one can then create the similar-looking FTS shown in Figure 1 with initial state a_Z. Here, an abstract state a_i corresponds to automaton state q_i, the focus and defocus steps are labeled with F^-, F^+ respectively, and the propositions known to be true at an abstract state are written next to that state.

This discussion showing the correspondence of automata and abstract FTS's gives a sketch of how the completeness and maximal model results are obtained. The following sections describe this process, as well as the soundness guarantees, in more detail.

3.1. Preliminaries

As discussed previously, it is easier to make the connection from properties to abstract systems if the properties are expressed as alternating tree automata. The satisfaction relation for automaton properties, $\models$, is conveniently defined in terms of infinite games. For uniformity, we formulate the abstraction relation, $\sqsubseteq$, also in terms of games. This enables us to prove the soundness of abstraction by combining winning strategies for the games showing that $M \sqsubseteq N$ and $N \models A$ into a winning strategy for the $M \models A$ game.

Infinite Games. We fix some terminology regarding infinite, two-player games. The players are called I and II. A game is played with a set of *configurations*. This set is partitioned into two subsets, one for each player. Some configurations are *initial*. An initialized sequence of configurations is one where the first element is initial. The *acceptance condition* of a game is a set of initialized sequences of configurations where, if a finite sequence is accepted, so is every infinite extension of this sequence. A *strategy* for player K (K $\in \{\mathrm{I}, \mathrm{II}\}$) is a partial function mapping a finite, initialized sequence of configurations that ends in a configuration for player K, into a new configuration. Only some strategies are allowed for a game; the restrictions are usually specified through a set of rules. Given strategies for both players, a *play*, π, of the game is an initialized sequence of configurations where for each $i \geq 0$, π_{i+1} is obtained by applying the appropriate players' strategy to the sequence $\pi_0 \ldots \pi_i$. A play is a win for player I if it satisfies the acceptance condition, and a win for player II otherwise. A strategy is *winning* for player K if every play of the game generated by following this strategy for player K (regardless of the strategy of the other player) is a win for player K.

Properties as Alternating Tree Automata. We represent branching time properties by *alternating tree automata* over the set AP of atomic propositions. An automaton, A, is given by a tuple $(Q, \hat{q}, \delta, \Theta)$, where Q is a finite set of states, $\hat{q} \in Q$ is the initial state, δ is a transition relation, and Θ is an *acceptance* condition, which is a set of infinite sequences over Q that is closed under the addition and removal of finite prefixes, and under finite stuttering (state repetition). We use a simple normal form for the transition function δ, which maps an automaton state q to one of the following forms — here, P is a proposition in AP, r is an automaton state, and R is a non-empty subset of Q: $P \mid \neg P \mid \bigwedge R \mid \bigvee R \mid \mathsf{EX}\, r \mid \mathsf{AX}\, r$. In the case of $\bigwedge R$ or $\bigvee R$, the states in R are the successors of q, and in the case of $\mathsf{EX}\, r$ or $\mathsf{AX}\, r$, r is. This successor relation should have no cycles which go only through the $\bigwedge$ and $\bigvee$ operators.

Automata of this type can encode logics commonly used to state correctness properties of programs, such as linear temporal logic, CTL, CTL*, and the propositional mu-calculus. A mu-calculus formula in positive normal form with no free variables and distinctly named bound variables corresponds directly to such an automaton [14]: each subformula is an automaton state, and the transition at that state is given by the subformula structure, which is one of the forms considered above. The acceptance condition is obtained by analyzing the fixpoint alternation in the formula. By changing EX to $\langle a \rangle$ and AX to $[a]$, where a is an edge label, we obtain automata that can encode the modal mu-calculus. We use the simpler form for clarity.

3.2. Focused Transition Systems

A *focused transition system* (FTS for short) is a simple extension of the mixed transition systems defined in Section 2. It is a tuple $(S, I, T^-, T^+, L^-, L^+, F^-, F^+, \Phi)$ where S is a set of *states*, I is a non-empty subset of *initial* states, $T^-, T^+ \subseteq S \times S$ are under- and over-approximate *transition relations*, respectively (the *must* and *may* relations of mixed transition systems), $L^-, L^+ : S \to 2^{AP}$ are functions giving, for each state s, lower and upper bounds on the *propositions* known to hold at s (so that $L^-(s) \subseteq L^+(s)$ holds for all s), $F^-, F^+ \subseteq S \times 2^S$ are *focus* and *de-focus* relations, respectively, such that for every s, $\{s\}$ belongs to $F^-(s)$ and to $F^+(s)$, and Φ is a *fairness constraint*, a set of infinite sequences over S that is invariant under the addition and removal of finite prefixes, and under finite stuttering. We sometimes use curried forms of the relations: for instance, $F^-(s) = \{X \mid F^-(s, X)\}$.

An *ordinary transition system* (TS for short) is obtained by setting $T^+ = T^-$, $L^+ = L^-$, F^+ and F^- so that they relate each state s to only $\{s\}$, and Φ to the set of all infinite sequences over S (i.e., every sequence is fair).

3.3. The Model Checking Game for FTS's

The acceptance of an FTS $M = (S, I, T^-, T^+, L^-, L^+, F^-, F^+, \Phi)$ by an automaton $A = (Q, \hat{q}, \delta, \Theta)$, both spec-

At a configuration (s, q), based on the form of $\delta(q)$:

- P: player I wins if, and only if, $P \in L^-(s)$.
- $\neg P$: player I wins if, and only if, $P \notin L^+(s)$.
- $\vee R$: Player I picks a focus set, X, from $F^-(s)$; Player II picks a state u in X; then Player I picks r from R. The next configuration is (u, r).
- $\wedge R$: Player I picks a defocus set, X, from $F^+(s)$; Player II picks r from R; Player I picks u from X. The next configuration is (u, r).
- $\mathsf{EX}\, r$: Player I picks t such that $T^-(s, t)$ holds, the next configuration is (t, r).
- $\mathsf{AX}\, r$: Player II picks t such that $T^+(s, t)$ holds, the next configuration is (t, r).

Figure 2. The FTS Model Checking Game

ified over atomic proposition set AP, is determined by a game. The configurations are pairs of the form (s, q), where s is a state of M and q is an automaton state. Roughly speaking, at such a configuration, player I tries to show that the property represented by state q holds at s, while player II tries to refute this claim. Other intermediate configurations are also generated as part of the moves. Note that although there are no draws possible in the game, if player I does not succeed to show that q holds at s, this does not imply that player II has now demonstrated that $\neg q$ holds at s. In this sense, the setting is 3-valued. The moves of the game, which are based on the automaton transition relation δ, are specified in Figure 2.

A short explanation of the rules follows. The P ($\neg P$) rule ensures that a proposition P (resp. $\neg P$) holds iff P is in the lower (resp. outside the upper) bound of the propositional labeling. The $\vee$ and the $\wedge$ rules strictly generalize the normal interpretation of these operators. In the normal interpretation, a state s satisfies $\vee R$ if, and only if, s satisfies some r in R. This is modeled by player I picking $X = \{s\}$ in its move. However, the rule allows another way of satisfying the disjunction: by splitting s into cases (focusing), and showing that each case satisfies some r in R. This is modeled by player I picking some other X in its move, after which player II checks that every case (its choice of u) satisfies some r (player I's choice). In the $\wedge$ rule, the extension is similar. In the normal interpretation, a state s satisfies $\wedge R$ if, and only if, s satisfies all r in R. This is modeled by player I picking $X = \{s\}$ in its move. However, the rule allows another way of satisfying the conjunction: by generalizing s into a set of more abstract states (defocusing), and showing that each r in R holds for some state in this set. This is modeled by player I picking some other X from $F^+(s)$ in its move, after which player II picks an arbitrary r in R, and player I chooses an appropriate abstract state u from X. For ordinary transition systems, only the normal interpretation applies, as expected, since $F^-(s)$ and $F^+(s)$ do not contain a choice other than $\{s\}$. The $\mathsf{EX}\, r$ rule is interpreted relative to the under-approximate (T^-) relation, while in the $\mathsf{AX}\, r$ rule, player II picks an over-approximate successor of s in trying to refute the claim that all successors of s satisfy r. We refer to a move specified by the rule for the EX (AX) operator as an EX (AX) move.

The initial configurations of this game are pairs from $I \times \{\hat{q}\}$. A finite play of the game is a win for player I (II) as stated in the first two rules. An infinite play π is a win for player I if it either fails to satisfy the fairness requirement for M, or it satisfies the acceptance condition of A; i.e., letting π_M, π_A be π's projections on the states of M and A respectively, if $\Phi(\pi_M) \Rightarrow \Theta(\pi_A)$ holds of the play. We say that FTS M *satisfies* the automaton property A (written as $M \models A$), iff player I has a winning strategy in this game from all initial configurations.

3.4. Abstraction between FTS's

We now define a notion of abstraction between FTS's M and N through a game. Normally, an abstraction relation (e.g., simulation) is specified by a co-inductive definition. It is easier to state the winning condition with the game-theoretic form, because a play of the game induces a joint computation of the two processes, and it is then possible to relate the individual fairness constraints on this joint computation. The approaches are equivalent for trivial fairness conditions (cf. [42]).

The configurations of the game are elements of $S_M \times S_N$. The initial configurations are elements of $I_M \times I_N$. Informally speaking, at a configuration (s, t), player I tries to show that t abstracts s, while player II tries to refute this claim. The game moves are shown in Figure 3. Intermediate configurations of a different form may be created in these moves.

This game is quite similar in its zig-zag nature to the bisimulation game (cf. [42]) with – of course – special rules for the focus operations. Informally speaking, the first two rules ensure that the propositional labeling is more precise for s than t: propositions definitely true (false) at t are definitely true (false) at s. The last two rules ensure that the T_N^- (T_N^+) transition relation under(over)-approximates the T_M^- (T_M^+) relation.

To gain further intuition for the focus rules, think of the game as defining an abstraction relation $\rho \subseteq S_M \times S_N$, where $\rho(s, t)$ iff player I has a winning strategy from configuration (s, t). Let the concretization of t, $\gamma(t)$, be $\{s \in S_M \mid \rho(s, t)\}$ for any $t \in S_N$. Consider the case that M

At a configuration (s,t), where $s \in S_M$, and $t \in S_N$, each of the following rules, when applicable, defines the possible moves:

- L^- labeling: Player II chooses P from $L_N^-(t)$; Player I wins iff P is in $L_M^-(s)$.
- L^+ labeling: Player II chooses P that is not in $L_N^+(t)$; Player I wins iff P is not in $L_M^+(s)$.
- F^- transitions: Player II picks a set Y from $F_N^-(t)$; Player I responds with a set X from $F_M^-(s)$; Player II picks an element x from X; Player I responds with an element y from Y; the next configuration is (x,y).
- F^+ transitions: Player II picks a set Y from $F_N^+(t)$; Player I responds with a set X from $F_M^+(s)$; Player II picks an element y from Y; Player I responds with an element x from X; the next configuration is (x,y).
- T^- transitions: Player II chooses a state v such that $T_N^-(t,v)$ holds; Player I responds with u such that $T_M^-(s,u)$ holds; the next configuration is (u,v).
- T^+ transitions: Player II chooses a state u such that $T_M^+(s,u)$ holds; Player I responds with v such that $T_N^+(t,v)$ holds; the next configuration is (u,v).

Figure 3. The FTS Abstraction Game Moves

is an ordinary TS. Suppose t is such that $F^-(t)$ contains a set Y other than $\{t\}$; the states in Y define a case-split of t in the following sense. In the focus rule, as M is ordinary, the only possible choice for player I in response to player II's choice of Y is to take $X = \{s\}$. So the only choice for x is s, after which player II chooses y from Y. As (s,y) must be a winning configuration, we have $s \in \gamma(y)$. Letting $\gamma(Y) = (\bigcup y : y \in Y : \gamma(y))$, we obtain that s is in $\gamma(Y)$. As this is true for every s in $\gamma(t)$, it follows that $\gamma(t) \subseteq \gamma(Y)$, and this is the case for each focus set Y of t. Thus, loosely speaking, $\gamma(t)$ is split up (i.e., focused) into the concretizations of the states in Y, which together cover $\gamma(t)$. Reasoning in a similar manner for the F^+ rule, one obtains that $\gamma(t) \subseteq (\bigcap y : y \in Y : \gamma(y))$, for each defocus choice Y of t. Therefore, the concretization of each abstract state in Y generalizes (de-focuses) the concretization of t.

A finite play of the game is a win for player I only as stated in the first two rules. Any infinite play with only F^-, F^+ moves from some point on is a win for player I. Let π be any other type of infinite play, and let π_M, π_N be its projections on the states of M and N, respectively. The play π is a win if $\Phi_M(\pi_M) \Rightarrow \Phi_N(\pi_N)$ holds for it. We say that N *abstracts* M (or M *refines* N, both written as $M \sqsubseteq N$) iff player I has a *winning* strategy in this game, played from the initial configurations, which are pairs from $I_M \times I_N$. The following theorem is a simple consequence of the definitions.

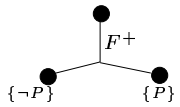

Figure 4. An Inconsistent FTS

Theorem 5 *The abstraction relation is a pre-order.*

Note that it is quite possible to define FTS's that are *inconsistent*, in the sense that the defined FTS does not abstract any ordinary transition system (in this regard, FTS's are like automata). Perhaps the simplest example is the FTS shown in Figure 4; any TS that refines it must satisfy both P and $\neg P$ in its initial state, which is impossible.

3.5. Soundness Theorem

In this section we state the main soundness theorem that relates $\sqsubseteq$ and $\models$. The theorem has an implicit 3-valued nature: if $M \sqsubseteq N$ and $N \models A$, then $M \models A$; however, if N does not satisfy A, whether M satisfies A cannot be resolved. To do so requires a distinct check of whether $N \models \neg A$. I.e., for abstract models, $N \models A$ being false is not equivalent to $N \models \neg A$ being true — there is a "gap". Bruns and Godefroid define in [5] a single, 3-valued model checking game that takes the place of these two checks; our satisfaction game can be extended in a similar manner. We do not do so here in order to concentrate attention on the role played by the focusing operations.

Theorem 6 ***[Soundness]*** *For all FTS's N, M, and every automaton property A, if $M \sqsubseteq N$ and $N \models A$, then $M \models A$.*

The proof of this theorem proceeds as follows. As $M \sqsubseteq N$ and $N \models A$, there are winning strategies for player I in both games. A winning strategy for player I in the $M \models A$ game can be crafted by combining the given winning strategies. The new strategy is defined through induction on the length of plays. The inductive hypothesis associates each partial play obtained through the new strategy with partial plays obtained with the given winning strategies; the associated plays are extended further in the induction step.

3.6. The Completeness Theorem

Theorem 7 ***[Completeness]*** *For every FTS M and automaton property A such that $M \models A$, there is a finite FTS N such that $M \sqsubseteq N$ and $N \models A$.*

The abstract transition system N is essentially the automaton A itself, suitably re-structured to form an FTS as indicated by the worked-out example in this section; we denote this FTS by $\overline{A}$. It is constructed as follows. Let M be the FTS from the statement of the theorem, $A = (Q, \hat{q}, \delta, \Theta)$, and $\mathcal{G}$ be the game showing that $M \models A$. We represent an abstract state derived from an automaton state q by $\overline{q}$. Then, $\overline{A} = (S, I, T^-, T^+, L^-, L^+, F^-, F^+, \Phi)$, where the components are specified as follows.

For a subset of automaton states R, let $\overline{R} = \{\overline{q} \mid q \in R\}$. The set of states of the FTS, S, is $\overline{Q}$ together with a distinct state $\top$. The $\top$ state is the most abstract state possible; it is used as a target for T^+ transitions. The single initial state is $\overline{\hat{q}}$. For an abstract state $\overline{q}$ where $\delta(q) = P$ ($P \in AP$), let $L^-(\overline{q}) = \{P\}$ and $L^+(\overline{q}) = AP$; i.e., only P is definitely true at $\overline{q}$, the truth value of other propositions is unknown. If $\delta(q) = \neg P$, let $L^-(\overline{q}) = \emptyset$ and $L^+(\overline{q}) = AP \backslash \{P\}$; i.e., P is definitely false at $\overline{q}$, the truth value of other propositions is unknown. At all other states r, let $L^-(r) = \emptyset$, and $L^+(r) = AP$; i.e., the truth value of all propositions is unknown.

An abstract state $\overline{q}$ where $\delta(q) = \mathsf{EX}\, r$ has a T^- transition to $\overline{r}$; there are no other T^- transitions. An abstract state $\overline{q}$ where $\delta(q) = \mathsf{AX}\, r$ has a T^+ transition to $\overline{r}$; all other states (including $\top$) have a T^+ transition to $\top$.

The F^- and F^+ relations for a state s include $\{s\}$, as required. Furthermore, for an abstract state $\overline{q}$ where $\delta(q) = \vee R$, there is an additional focus choice: the set $\overline{R}$, and for an abstract state $\overline{q}$ where $\delta(q) = \wedge R$, there is an additional defocus choice: the set $\overline{R}$. Let $\overline{\Theta}$ be the set of infinite sequences over S obtained by replacing each state q in a sequence in Θ with its counterpart $\overline{q}$. The fairness condition Φ is the union of $\overline{\Theta}$ with all infinite sequences that from some point on have only the $\top$ state.

To prove the theorem, we show that $M \sqsubseteq \overline{A}$ (abstraction), and that $\overline{A} \models A$ (model checking). We use the winning strategy in $\mathcal{G}$ as a guide for formulating winning strategies in the abstraction and model checking games for $\overline{A}$.

Abstraction: The strategy for the abstraction game is essentially identical to the winning strategy for $\mathcal{G}$, with minor alterations needed to account for the changes in going from A to $\overline{A}$. Inductively, player I tries to ensure that, for every partial play π of the game, either the play is "stuck" in the $\top$ state, or there is a stuttering-equivalent partial play π' of $\mathcal{G}$ generated by its winning strategy. This partial play is extended in the induction step. The winning condition for a play is $\Phi_M \Rightarrow \Phi$. If a play π gets stuck in the $\top$ state, it is a win trivially, since Φ is true. Otherwise, as π is stuttering-equivalent to a winning play π' of $\mathcal{G}$, since $(\Phi_M \Rightarrow \Theta)$ holds for π', $(\Phi_M \Rightarrow \overline{\Theta})$ holds of π, so that it is a win for the abstraction game.

Model Checking: In the model checking game, player I can ensure that, for every play of the game, the only configurations that occur are those labeled with $(\overline{q}, q)$, for $q \in Q$. This can be ensured for the initial configuration, which is labeled with $(\overline{\hat{q}}, \hat{q})$, and it is quite simple to maintain this invariant when extending a play. Thus, plays never enter the $\top$ state, so the winning condition of the game, which is $\Phi \Rightarrow \Theta$, reduces to $\overline{\Theta} \Rightarrow \Theta$. Since this is true trivially by the invariant, every play results in a win for Player I.

Corollary 1 ***[Completeness for Ordinary TS's]*** *For every ordinary transition system M and automaton property A such that $M \models A$, there is a finite FTS N such that $M \sqsubseteq N$ and $N \models A$.*

Recall that a maximal model for a property A is a model of which all other models of A are refinements.

Corollary 2 ***[Maximal Model Theorem]*** *For any automaton property A, $\overline{A}$ is a maximal model.*

4. Discussion

By focusing (!) on the completeness question for branching time logics, we have uncovered an interesting new class of transition systems to use for abstraction, with strong links to alternating tree automata and deductive proof methods. Space constraints preclude a discussion of implementation issues; however, we would like to point out that the machinery necessary for defining focus operators is already present in abstraction methods that use Galois connections (cf. [8, 9]), since the lattice structure on the abstract domain can be used to guide the definition of *focus* and *de-focus* of abstract states.

Acknowledgements. Glenn Bruns brought the small model theorem to our attention. This work is supported in part by NSF grant CCR-0341658.

References

[1] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR*, vol. 1466 of *LNCS*, 1998.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *COMPOS*, vol. 1536 of *LNCS*, 1997. (full version in J.ACM 49(5), 2002).

[3] M. Browne, E. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Journal of Theoretical Computer Science*, 59:115–131, 1988.

[4] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *CAV*, vol. 1633 of *LNCS*, 1999.

[5] G. Bruns and P. Godefroid. Temporal logic query checking. In *LICS*, 2001.

[6] E. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, vol. 131 of *LNCS*, 1981.

[7] R. Cleaveland, S. P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In *SAS*, vol. 983 of *LNCS*, 1995.

[8] 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.

[9] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems. *TOPLAS*, 19(2), 1997.

[10] D. R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, July 1996.

[11] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. LICS'04 (these proceedings).

[12] E. Emerson. Automata, tableaux and temporal logics (extended abstract). In *Logic of Programs*, vol. 193 of *LNCS*, 1985.

[13] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics*. Elsevier and MIT Press, 1990.

[14] E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, 1991.

[15] E. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *LICS*, 1986.

[16] E. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Sci. of Comp. Programming*, 8(3), 1987.

[17] E. A. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *CAV*, vol. 697 of *LNCS*, 1993.

[18] R. J. v. Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G. X. Ritter, editor, *Information Processing 89*, North-Holland, Amsterdam, 1989.

[19] P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Verification, Model Checking, and Abstract Interpretation*, vol. 2575 in LNCS, 2003.

[20] O. Grumberg and D. Long. Model checking and modular verification. *TOPLAS*, 1994.

[21] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1), 1985.

[22] R. Hossley and C. Rackoff. The emptiness problem for automata on infinite trees. In *FOCS*, 1972.

[23] M. Huth, R. Jagadeesan, and D. Schmidt. A domain equation for refinement of partial systems. *Math. Struct. in Comp. Science (to appear)*.

[24] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163(1), 2000.

[25] Y. Kesten, A. Pnueli, and M. Vardi. Verification by augmented abstraction: The automata-theoretic view. *JCSS*, 62(4), 2001.

[26] O. Kupferman and M. Vardi. Modular model checking. In *COMPOS*, vol. 1536 of *LNCS*, 1997.

[27] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching time model checking. *J. ACM*, 47(2), 2000.

[28] K. G. Larsen and B. Thomsen. A modal process logic. In *1988 IEEE Symposium on Logic in Computer Science*, TCMFC, Computer Society Press, Washington, 1988.

[29] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, 1990.

[30] M. Maidl. The common fragment of CTL and LTL. In *FOCS*, 2000.

[31] Z. Manna and A. Pnueli. How to cook a temporal proof system for your pet language. In *POPL*, 1983.

[32] P. Manolios. A compositional theory of refinement for branching time. In *CHARME*, 2003.

[33] P. Manolios and R. J. Trefler. Safety and liveness in branching time. In *LICS*, 2001.

[34] R. Milner. An algebraic definition of simulation between programs. In *2nd IJCAI*, 1971.

[35] K. S. Namjoshi. Certifying model checkers. In *CAV*, vol. 2102 of *LNCS*, 2001.

[36] K. S. Namjoshi. Abstraction for branching time properties. In *CAV*, vol. 2725 of *LNCS*, 2003.

[37] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, vol. 104 of *LNCS*, 1981.

[38] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

[39] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th International Symposium on Programming*, vol. 137 of *LNCS*, 1982.

[40] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *TOPLAS*, 24(3), 2002.

[41] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS*, volume 2988 of *LNCS*, 2004.

[42] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[43] T. Uribe. *Abstraction-Based Deductive-Algorithmic Verification of Reactive Systems*. PhD thesis, Stanford University, 1999.