Modular Generic Verification of LTL Properties for Aspects

Max Goldman Shmuel Katz
Computer Science Department
Technion — Israel Institute of Technology
{mgoldman, katz}@cs.technion.ac.il

ABSTRACT

Aspects are separate code modules that can be bound ("woven") to a base program at joinpoints to provide an augmented program. A novel approach is defined to verify that an aspect state machine will provide desired properties whenever it is woven over a base state machine that satisfies the assumptions of the aspect. A single state machine is constructed using the tableau of the linear temporal logic (LTL) description of the assumptions, a description of the joinpoints, and the state machine of the aspect code. A theorem is shown that if the constructed machine satisfies the desired properties, so will an augmented state machine using any base machine that satisfies the assumptions. The theorem is stated and shown for assumptions and properties given in LTL, for a somewhat restricted form of joinpoint description, and for aspect code that ends in states already reachable in the base state machine. A language-based description of aspects, as in AspectJ, can be converted to a state machine version using existing tools, thus providing generic modular verification of code-level aspects.

1. INTRODUCTION

1.1 Aspect-Oriented Programming

The aspect-oriented approach to software development is one in which concerns that cut across many parts of the system are encapsulated in separate modules called *aspects*. For example, when security or logging are encapsulated in an aspect, this aspect contains both the code associated with the concern, called *advice*, and a description of when this advice should run, called a *pointcut descriptor*. The pointcut descriptor identifies those points in the execution of a program at which the advice should be invoked. The combination of some *base program* with an aspect (or in general, a collection of aspects), is termed an *augmented program*.

1.2 Formal Verification

In this work we are concerned with generic formal verification of aspects relative to a specification. The specification of an aspect consists of *assumptions* about any base program to which the aspect can reasonably be woven, and *desired properties* intended to hold for the augmented program. We view both base programs and aspect code as nondeterministic finite state machines, in which particular computations are realized as infinite sequences of states within the machine. For both assumptions and desired properties to be verified we consider formulas written in linear temporal logic (LTL). An LTL formula consists of a *path formula* using *temporal quantifiers* and logical combinations of *atomic propositions*, prefixed by a single (usually implicit) *universal path quantifier*. The atomic propositions in a formula refer to the labels of states in a finite state machine; temporal quantifiers specify when these assertions about states must be true. The universal path quantifier requires that, in order for some initial state to satisfy an LTL assertion, all infinite paths from that state must satisfy the path formula. In general, a state machine also includes a fairness constraint, and only fair paths are considered.

1.3 Modular Aspectual Verification

It is clear that given a base program, a collection of aspects with their pointcut descriptors and advice, and a system for *weaving* together these components to produce a stand-alone augmented program, we can verify properties of this augmented system using the usual model checking techniques. Such weaving involves adding edges from joinpoint states of the base program to the initial states of the advice, and from the states after an advice segment to states where base program statements are executed.

It would be preferable, however, if we could employ a modular technique in which the aspect can be considered separately from the base program. This would allow us to:

- obtain verification results that hold for a particular aspect with any base program from some class of programs, rather than for only one base program in particular;
- use the results to reason about the application of aspects to base programs with multiple evolving state machines describing changing configurations during execution, or to other base systems not amenable to model checking; and
- avoid model checking augmented systems, which may be significantly larger than their base systems, and whose unknown behavior may resist abstraction.

The second point above relates to general object-oriented programs that create new instances of classes (objects) with associated state machine components. Often, the assumption of an aspect about the key properties of those base state machines to which it may be woven can indeed be shown to hold for every possible machine that corresponds to an object configuration of a program. For example, it may involve a so-called *class invariant*, provable by reasoning directly on class declarations, as in [1]. This point and more details on the connections between code-based aspects (as in AspectJ) and the state machine versions seen here are discussed in Section 5.

This problem of creating a single generic model that can represent any possible augmented program for an aspect woven over some class of base programs is especially difficult because of the aspect-oriented notion of *obliviousness*: base programs are generally unaware of aspects advising them, and have no control over when or how they are advised. There are no explicit markers for the transfer of control from base to advice code, nor are there guarantees about if or where advice will return control to the base program.

1.4 Results

In this paper we show how to verify once-and-for-all that for any base state machine satisfying the assumptions of the aspect, and for a weaving that adds the aspect advice as indicated in the joinpoint description, the resulting augmented state machine is guaranteed to satisfy the desired properties given in the specification. A single generic state machine is constructed from the tableau of the assumption, the pointcut descriptor, and the advice state machine, and verified for the desired property. Then, when a particular base program is to be woven with the aspect, it is sufficient to establish that the base state machine satisfies the assumption. Thus the entire augmented program never has to be model checked, achieving true modularity and genericity in the proof. This approach is especially appropriate for aspects intended to be reused over many base programs, e.g., those in libraries or middleware components.

LTL model checking is based on creating a tableau state machine automaton that accepts exactly those computations that satisfy the property to be verified. Usually, the negation of this machine is then composed as a cross-product with the model to be checked. A counter-example is produced when the composed system contains some infinite path, and the property is satisfied for the model when the cross-product has no such paths. Here we use the tableau of the assumption in a unique way, as the basis of the generic model to be checked for the desired property. It represents any base machine satisfying the assumption, because the execution sequences of the base program can be abstracted by sequences in the tableau.

For the soundness theorem presented in Section 4, the aspects treated are assumed to be *weakly invasive*, as defined in [7]. This means that when advice has completed executing, the system continues from a state that was already reachable in the original base program (perhaps for different inputs or actions of the environment). Many aspects fall into this category, including *spectative* aspects that never modify the state of the base system (logging is a good example), and *regulative* aspects that only restrict the reachable state space (for example, aspects implementing security checks). Also weakly invasive would be an aspect to enforce transactional requirements, which might roll back a series of changes so that the system returns to the state it was in before they were made. Even a 'discount policy' aspect that reduces the price on certain items in a retail system is weakly invasive, since the original price given as input could have been the discounted one.

Additionally, we assume that any executions of an augmented program that infinitely often include states resulting from aspect advice will be fair (and thus must be considered for correctness purposes). The version here does not treat multiple aspects or joinpoints influenced by the introduction of advice, although the approach can be expanded to treat such cases as well.

In the following section, needed terms and constructs are defined. Section 3 presents the algorithm, and Section 4 gives a proof of soundness in the weakly invasive aspect case. This section also gives an example. Section 5 details works related to the result here, and is followed by the conclusion.

2. DEFINITIONS

2.1 LTL Tableaux

Intuitively, the tableau of an LTL forumla f is a state machine whose fair infinite paths are exactly all those paths which satisfy the formula f. This intuition will be realized formally in Theorem 1 below.

In the context of performing model checking to verify satisfaction of an LTL property, a tableau is constructed for the *negation* of that property, in order to capture all possible computations that would cause a machine *not* to satisfy the formula in question. It is important to stress that here we use the tableau for the original non-negated formula. Nevertheless, because of the use of tableaux by LTL model checking tools, modules to perform the construction of a formula's tableau are available. For exploratory purposes, the authors have used the translator module of NuSMV [10], which produces a (tableau) finite state machine from a given LTL formula.

We define T_f, the tableau for LTL path formula f, as given in *Model Checking* [3] in the section on "Symbolic LTL Model Checking" (6.7). In this construction, the original formula is decomposed into the set of *elementary formulas* it contains, where all other temporal operators, such as *from now on* (G) and *eventually* (F), are expressed in terms of *next* (X) and *strong until* (U). Each state in the tableau is a subset of these elementary formulas, and the path relation between these states is defined by means of a function $sat(g)$, which captures the set of states in which subformula g of f is satisfied.

We denote $T_f = (S_T, S_0^T, R_T, L_T, F_T)$, where S_T is the set of states; S_0^T is the set of initial states, R_T is the transition relation, L_T is the labeling function, and F_T is the set of fair state sets. For ease of discussion, we clarify the definition as follows:

Define S_0^T, where for $\chi = Af$, we have $T_f \models \chi$:

$$S_0^T = sat(f)$$

Define F_T, where any fair path in T_f must visit each set in F_T infinitely many times:

$$F_T = \{sat((\neg(g \,\mathsf{U}\, h)) \vee h) \mid g \,\mathsf{U}\, h \text{ is a subformula of } f\}$$

This fairness constraint guarantees that obligations of the form $g \,\mathsf{U}\, h$ are fulfilled, either by visiting a state in $sat(h)$ infinitely often, or by infinitely often visiting a state outside of $sat(g \,\mathsf{U}\, h)$, which can only be reached by going via $sat(h)$ according to the construction of the path relation (not detailed here).

Two notable properties of T_f will be used below. First, if AP_f is the set of atomic propositions in f, then $L_T : S \to \mathcal{P}(AP_f)$ — that is, the labels of the states in the tableau will include sets of the atomic propositions appearing in f. A state in any machine is given a particular label if and only if that atomic proposition is true in that state.

The second interesting feature is a main theorem from the discussion in [3]:

Definition 1. For path π, let $label(\pi)$ be the sequence of labels (subsets of AP) of the states of π. For such a sequence $l = l_0, l_1, \ldots$ and set Q, let $l|_Q = m_0, m_1, \ldots$ where for each $i \geq 0$, $m_i = l_i \cap Q$.

THEOREM 1. *Given T_f, for any Kripke structure M, for all fair paths π' in M, if $M, \pi' \models f$ then there exists fair path π in T_f such that π starts in S_0^T and $label(\pi')|_{AP_f} = label(\pi)$.*

That is, for any possible computation of M satisfying formula f, there is a path in the tableau of f which matches the labels within AP_f along the states of that computation.

In the algorithm of Section 3, we restrict the tableau to its reachable component. Such restriction does not affect the result of this theorem, since all reachable paths are preserved, but is necessary in order to achieve useful results. This follows from the observation that the tableau for the negation of a formula has precisely the same states and transition relation, but the complementary set of initial states. Thus, any unreachable portion of the tableau is liable to contain exactly those behaviors which violate the formula of interest.

Finally, for $\chi = Af$, define $T_\chi = T_f$ as a convenient notation (a tableau can only be constructed for a path formula).

2.2 Aspects

An aspect machine $A = (S_A, S_0^A, S_{ret}^A, R_A, L_A)$ over atomic propositions AP is defined as usual for a machine with no fairness constraint, with the following addition:

Definition 2. S_{ret}^A is the set of *return states* of A, where $S_{ret}^A \subseteq S_A$ and for any state $s \in S_{ret}^A$, s has no outgoing edges.

2.3 Pointcuts

We do not give a prescriptive definition for pointcut descriptors here; in practice pointcut descriptions might take a number of forms. However, we require that descriptors operate in the following manner:

Definition 3. Given a pointcut descriptor ρ over atomic propositions AP and a finite sequence l of labels (subsets of AP), we can ask whether or not the end of l is *matched* by ρ, written $l \models\!\!\!| \rho$.

A reasonable choice for describing pointcuts might be LTL path formulas containing only past temporal operators. For example, the descriptor $\rho_1 = a \wedge \mathsf{Y}\, b \wedge \mathsf{Y}\, \mathsf{Y}\, b$ would match sequences ending with a state where a is true, preceded by b, preceded by another b (operator Y is the past analogue of X). Other languages could be imagined, for example regular expressions, where $\rho_2 = \mathsf{true}^* \cdot b \cdot b \cdot a$ might be equivalent to ρ_1. The use of regular expressions over automata is popular in industrial specification languages and has been examined in formal combination with LTL for example in [2].

2.4 Specifications

In addition to its advice, in the form of state machine A, and pointcut, described by ρ, an aspect is considered to have two pieces of formal specification:

- Formula ψ expresses the assumptions made by the aspect about any base machine to which it will be woven. This ψ is thus a requirement to be met by any such machine.
- Formula ϕ expresses the desired result to be satisfied by any augmented machine built by weaving this aspect with a conforming base machine. In other words, ϕ is the guarantee of the aspect.

2.5 Weaving

Weaving is the process of combining a base machine with some aspect according to a particular pointcut descriptor; the result is an augmented machine that includes the advice of the aspect.

The weaving algorithm has the following inputs:

- aspect machine $A = (S_A, S_0^A, S_{ret}^A, R_A, L_A)$ over AP,
- pointcut ρ over AP, and
- base machine $B = (S_B, S_0^B, R_B, L_B, F_B)$ over $AP_B \supseteq AP$.

And it produces as output:

- augmented machine $\widetilde{B} = (S_{\widetilde{B}}, S_0^{\widetilde{B}}, R_{\widetilde{B}}, L_{\widetilde{B}}, F_{\widetilde{B}})$.

The weaving is performed in two steps. First we construct from the base machine B a new state machine B^ρ which is *pointcut-ready* for ρ, wherein each state either definitely is or is not matched by ρ. Then we use B^ρ and A to build the final augmented machine $\widetilde{B}$.

This two-step division of the weaving process means that the algorithm cannot handle a number of problematic cases: when the pointcut descriptor matches advice states, and thus advice should be inserted on other advice; when the addition of advice states creates a new matching pointcut in the computation, and advice should be inserted; and when the addition of advice states causes a location that once matched a pointcut selector not to match it any longer. Proper handling of these scenarios is the subject of ongoing work.

2.5.1 Constructing a Pointcut-Ready Machine

Pointcut-ready machine $B^\rho = (S_{B^\rho}, S_0^{B^\rho}, R_{B^\rho}, L_{B^\rho}, F_{B^\rho})$ is a machine in which unwinding of certain paths has been performed, so that we can separate paths which match pointcut descriptor ρ from those that do not. The pointcut-ready machine contains states with a new label, *pointcut*, that indicates exactly those states where the descriptor has been matched.

This machine must meet the following requirements:

- $S_{B^\rho} \supseteq S_B$
- $S_0^{B^\rho} = S_0^B$
- L_{B^ρ} is a function from S_{B^ρ} to $\mathcal{P}\left(AP_B \cup \{pointcut\}\right)$
- For all finite-length paths $\pi = s_0, \ldots, s_k$ in B^ρ such that $s_0 \in S_0^{B^\rho}$, $label(\pi) \models \rho \Leftrightarrow s_k \models pointcut$.
- For all infinite sequences of labels $l = (\mathcal{P}(AP_B))^\omega$, there is a fair path π_{B^ρ} in B^ρ where $label(\pi_{B^\rho})|_{AP_B} = l$ if and only if there is a fair path π_B in B where $label(\pi_B) = l$.

Note that since B and B^ρ have the same paths (over AP, ignoring the added *pointcut* label), they must satisfy exactly the same LTL formulas over AP.

Figure 1 shows a simple example of this construction. Note that in state diagrams, the absence of an atomic proposition indicates that the proposition does not hold, not that the value is unknown or irrelevant. This is in contrast to a formula, where unmentioned propositions are not restricted.

2.5.2 Constructing an Augmented Machine

We construct the components of augmented machine $\widetilde{B} = (S_{\widetilde{B}}, S_0^{\widetilde{B}}, R_{\widetilde{B}}, L_{\widetilde{B}}, F_{\widetilde{B}})$ as follows:

- $S_{\widetilde{B}} = S_{B^\rho} \cup S_A$
- $S_0^{\widetilde{B}} = S_0^{B^\rho}$

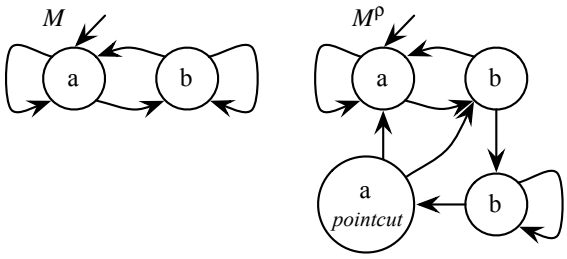

Figure 1: Constructing a pointcut-ready machine M^ρ for the given M and LTL past formula pointcut descriptor $\rho = a \wedge \mathbf{Y}\, b \wedge \mathbf{Y}\,\mathbf{Y}\, b$.

- $(s,t) \in R_{\widetilde{B}} \Leftrightarrow$

$$\begin{cases} (s,t) \in R_{B^\rho} \ \wedge \ s \not\models pointcut & \text{if } s,t \in S_{B^\rho} \\ (s,t) \in R_A & \text{if } s,t \in S_A \\ s \models pointcut \ \wedge \ t \in S_0^A & \\ \qquad \wedge\, L_{B^\rho}(s)|_{AP} = L_A(t) & \text{if } s \in S_{B^\rho},\ t \in S_A \\ s \in S_{ret}^A \wedge L_A(s) = L_{B^\rho}(t)|_{AP} & \text{if } s \in S_A,\ t \in S_{B^\rho} \end{cases}$$

Note that this relationship is 'if and only if.' In words, the path relation contains precisely all the edges from the pointcut-ready base machine B^ρ and from aspect machine A, except that *pointcut* states in B^ρ have edges only to matching start states in A, and aspect return states have edges to all matching base states.

- $L_{\widetilde{B}}(s) = \begin{cases} L_{B^\rho}(s) & \text{if } s \in S_{B^\rho} \\ L_A(s) & \text{if } s \in S_A \end{cases}$
- $F_{\widetilde{B}} = F_{B^\rho} \times S_A$

That is, $F_{\widetilde{B}} = \{F_i \cup S_A \mid F_i \in F_{B^\rho}\}$. A path is fair if it either satisfies the fairness constraint of the pointcut-ready machine, or if it visits some aspect state infinitely many times — a conservatively inclusive definition.

A weaving is considered *successful* if every reachable node in $S_{\widetilde{B}}$ has a successor according to $R_{\widetilde{B}}$.

2.6 Weakly Invasive Aspects

As mentioned above, we show our result for the broad class of aspects which, when they return from advice, do so to a reachable state in the base machine. Without this restriction, the aspect may return to unreachable parts of the base machine whose behavior is not bound by assumption formula ψ. In this case, the augmented system contains portions with unknown behavior, and is difficult to reason about in a modular way.

Definition 4. An aspect A and pointcut ρ are said to be *weakly invasive* for a base machine B if, for all states in S_{B^ρ} that are reachable by a fair path in $\widetilde{B}$, those states were reachable by a fair path in B^ρ.

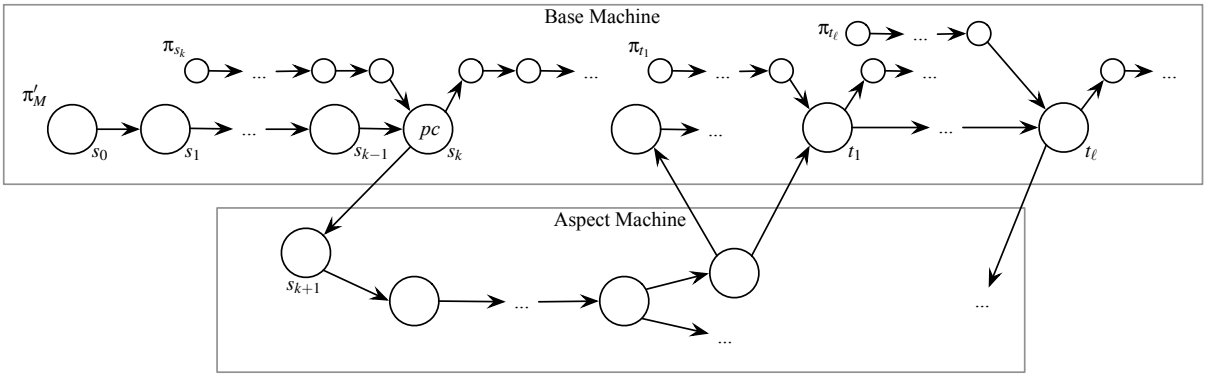

Figure 2: Using fair paths in M (small states along the top) to guarantee a matching path in $\widetilde{T_\psi}$.

In particular, this means that all states to which the aspect returns are states reachable in the pointcut-ready base machine. This could of course be checked directly, but would require construction of the augmented machine — precisely the operation we would like to avoid. In many cases, the aspect can be shown weakly invasive for any base machine satisfying its assumption ψ, by using static analysis, local model checking, or additional information (our reasoning in the discount price example from Section 1.4 uses such information). For further discussion, see [7].

3. ALGORITHM

The algorithm builds a tableau from ψ and weaves A with this tableau according to ρ, then performs model checking to verify the result with respect to ϕ. In the following section we prove that when this model check of the constructed augmented tableau succeeds, then for any base system satisfying requirement ψ, applying aspect A according to pointcut descriptor ρ will yield an augmented system satisfying result ϕ.

Given:

- set of atomic propositions AP;
- assumption ψ for base systems, an LTL formula over AP;
- desired result ϕ for augmented systems, an LTL formula over AP; and
- aspect machine A and pointcut descriptor ρ over AP.

Perform the following:

0. If it does not already, augment ψ with clauses of the form $\cdots \wedge (a \vee \neg a)$, such that ψ contains every atomic proposition $a \in AP$, without altering its meaning.
1. Construct T_ψ, the tableau for ψ. Since ψ contains every AP, the result of Theorem 1 will hold when all labels in AP are considered.
2. Restrict T_ψ to only those states reachable via a fair path.
3. Weave A into T_ψ according to ρ, obtaining $\widetilde{T_\psi}$.
4. Perform model checking in the usual way to determine if $\widetilde{T_\psi} \models \phi$.

4. CORRECTNESS

Given the components defined above, suppose that:

$$\widetilde{T_\psi} \models \phi \ .$$

What we have shown, then, is that the tableau for assumption ψ woven with aspect A according to ρ gives a resulting machine that satisfies desired augmented result ϕ. Our goal is to use the properties of $\widetilde{T_\psi}$ to show that A and ρ, when woven with *any* possible base machine M for which $M \models \psi$, will *always* yield an augmented $\widetilde{M}$ such that $\widetilde{M} \models \phi$. The proof below gives this result for a particular class of aspects.

THEOREM 2. *Given AP, ψ, ϕ, A, and ρ as defined, if $\widetilde{T_\psi} \models \phi$, then for any base program program M over a superset of AP such that A and ρ are weakly invasive for M, if $M \models \psi$ then $\widetilde{M} \models \phi$.*

PROOF. Since M and T_ψ contain exactly the same fair paths as M^ρ and T_ψ^ρ, and $M \models \psi$, by Theorem 1, for any fair path π_M in $M^\rho \models \psi$ starting from $S_0^{M^\rho}$, there is a fair path π_T in T_ψ^ρ with the same labels (restricted to AP). It suffices to show that after augmenting both of these pointcut-ready machines, this correspondence still holds.

Consider any fair path π'_M in $\widetilde{M}$ starting from an initial state.

Unmodified path Suppose no state on π'_M is labeled with *pointcut*. Then π'_M must be the same as some fair path π_M in M^ρ, which has matching fair path π_T in T_ψ^ρ. This path π_T contains no states labeled with *pointcut*, since for every finite subpath of π_M, ρ was not matched, and the labels on π_T are the same (restricted to AP).

Since π_T has no states labeled with *pointcut*, by the construction of $\widetilde{T_\psi}$, none of the edges along this path

have been removed during weaving. Therefore π_T is identical to a fair path π'_T in $\widetilde{T_\psi}$, and we have a matching path for π'_M.

Modified path Path π'_M must begin with a sequence of $k+1$ states $s_0, s_1, \ldots, s_k$ in M^ρ, where $k \geq 0$. Since s_k must be reachable from a fair path π_{s_k} in M^ρ, we can consider the path which begins $s_0, \ldots, s_k$ and continues along the remainder of π_{s_k} after s_k (see Figure 2). This path must also be fair, since it has the same infinite tail as π_{s_k} itself, and so must have a matching path in T^ρ_ψ; we begin π'_T by following this path.

Suppose that $s_k \models \mathit{pointcut}$, so s_{k+1} is in A. This state s_k can be labeled *pointcut* if and only if we have $\mathit{label}(s_0, \ldots, s_k) \models \rho$. In this case, the matching subpath in T^ρ_ψ must also match ρ, and will have *pointcut* on the state matching s_k. From both states, all edges go to machine A, so we can continue π'_T along an identical advice path; this includes the case when the advice never returns to the base machine.

If and when π'_M follows an edge from an aspect return state to a state t_1 in M^ρ, it does so to a state which is on a fair path π_{t_1} in that machine. There must be a matching path to π_{t_1} in the tableau. Furthermore, if we continue along a sequence of base machine states $t_1, \ldots, t_\ell$, since t_ℓ is also reachable from a fair path π_{t_ℓ}, the path which reaches t_1 via π_{t_1}, goes to t_ℓ, and then continues along π_{t_ℓ} from t_ℓ is also fair in M^ρ.

In $\widetilde{T_\psi}$, we have an edge from the aspect return state to every state whose labels match t_1; in particular, we must have an edge to the state corresponding to t_1 on the fair path matching our continuation from t_1 constructed above. We can continue the match π'_T for π'_M along this path.

If this continuation of base machine states is infinite, then the matching path in the tableau must be fair, since we are following the match of a fair path in M^ρ. If we never reach an infinite sequence of base states, but always reach another advice, then there must be some advice states which are visited infinitely many times, and again the path in the tableau is fair.

Therefore, for every fair path π'_M in $\widetilde{M}$ we have a corresponding fair path π'_T in $\widetilde{T_\psi}$. This correspondence completes the proof that $\widetilde{M} \models \phi$. □

4.1 Example

By way of example, suppose we have an aspect with base system assumption $\psi = \mathsf{A}\,\mathsf{G}\,((\neg a \wedge b) \rightarrow \mathsf{F}\,a)$ — that is, any state satisfying $\neg a \wedge b$ is eventually followed by a state satisfying a. We would like to prove that the application of our aspect to any base system satisfying ψ will give an augmented system satisfying result $\phi = \mathsf{A}\,\mathsf{G}\,((a \wedge b) \rightarrow \mathsf{X}\,\mathsf{F}\,a)$ — that is, any state satisfying $a \wedge b$ will eventually be followed by a later state satisfying a. While this example may not have clear correlation to a code-level problem, it serves to illuminate the capabilities of our technique.

Figure 3 shows the reachable portion of the tableau for the assumption ψ. In the diagram, shaded states are those contained in the only fairness set. The notation $\mathsf{X}g$, not actually

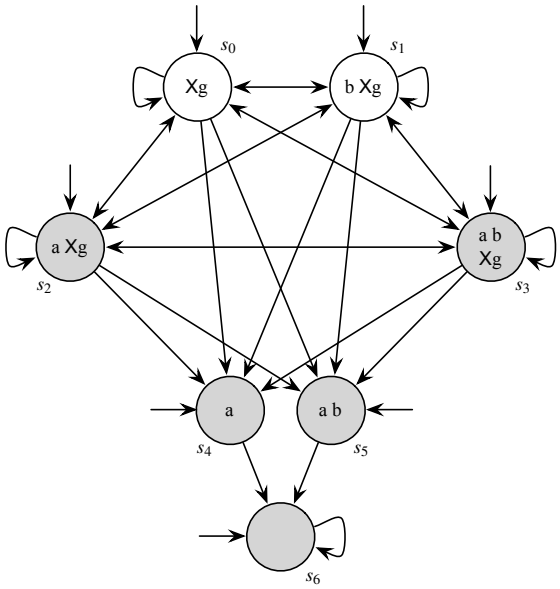

Figure 3: The reachable portion of tableau T_ψ for $\psi = \mathsf{A}\,\mathsf{G}\,((\neg a \wedge b) \rightarrow \mathsf{F}\,a)$.

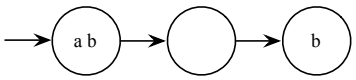

Figure 4: A simple aspect machine A.

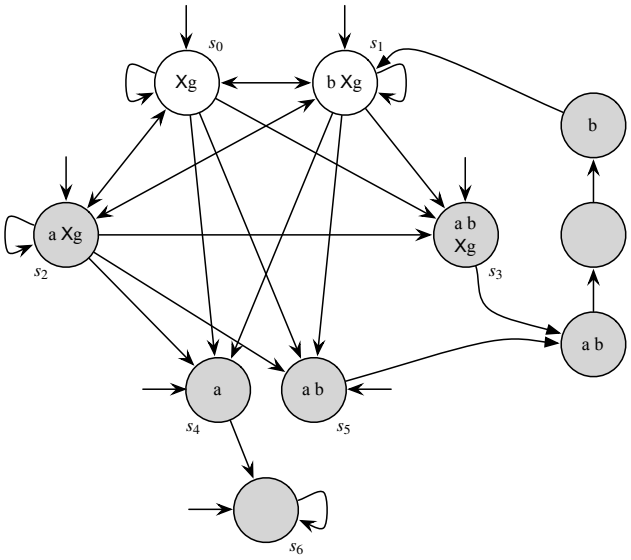

Figure 5: Augmented tableau $\widetilde{T_\psi}$, satisfying $\phi = \mathsf{A}\,\mathsf{G}\,((a \wedge b) \rightarrow \mathsf{X}\,\mathsf{F}\,a)$.

part of the state label, designates states in the tableau which satisfy $\mathsf{X}g$ for subformula $g = \mathsf{F}\,a$. For the example pointcut descriptor $\rho = (a \wedge b)$, this tableau machine is also pointcut-ready for ρ (since ρ references only the current state), simply by adding *pointcut* to the labels of s_3 and s_5.

Figure 4 shows the state machine A for the advice of our aspect. This advice will be applied at the states matched by ρ, and Figure 5 gives the weaving of A with T_ψ according to ρ. Model checking this augmented tableau will indeed establish that it satisfies the desired property ϕ. This result follows neither from the aspect nor base machine behavior directly, but from their combined behavior mediated by ρ. And since $\widetilde{T_\psi} \models \phi$, any $M \models \psi$ will yield $\widetilde{M} \models \phi$.

Reasoning intuitively about A and ρ without examining the tableau supports this conclusion: the advice is invoked at all states of such an M that match $(a \wedge b)$, the advice always leads to a state satisfying $(\neg a \wedge b)$, and ψ guarantees that from such a state we will always reach a state satisfying a, which is exactly the assertion of ϕ.

Figure 6 depicts a particular base machine M satisfying ψ, as could be easily verified by model checking. Again, the shaded states are those in the only fairness set. Although this M is small, it does contain atomic proposition c not 'visible' to the aspect, and it has a disconnected structure very much unlike the tableau. From Figure 7, one sees it is indeed the case that the augmented machine $\widetilde{M}$ satisfies ϕ — but there is no need to prove this directly by model checking. This holds true even though the addition of the aspect has made a number of invasive changes to M: state s_1 is no longer reachable, because its only incoming edge has been replaced by an advice edge; a new loop through s_0 has been added, when in M there was no path visiting s_0 more than once; there is a new path connecting the previously separated left-hand component to the right-hand; and so forth. In more realistic examples, the difference in size between the augmented tableau (involving only ψ, ρ, and A) and a concrete augmented system with advice over a full base machine would be substantial.

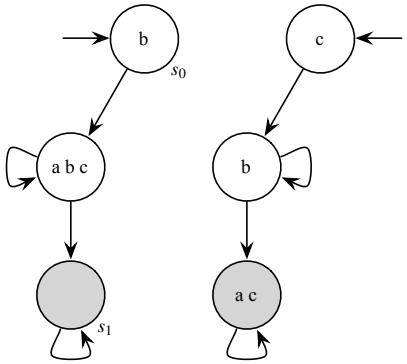

Figure 6: One particular base machine $M \models \psi$.

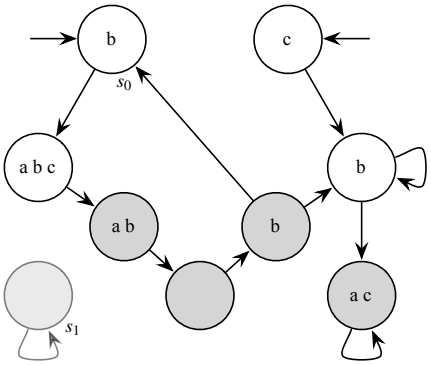

Figure 7: Augmenting M with A according to ρ gives result $\widetilde{M} \models \phi$.

5. RELATED WORK

The first work to separately model check the aspect state machine segments that correspond to advice is [9], where the verification is modular in the sense that base and aspect machines are considered separately. The verification method also allows for joinpoints within advice to be matched by a pointcut and themselves advised. However, the treatment there is for a particular aspect woven directly to a particular base program. Additionally, it shows only how to extend properties which hold for that base program, proving that the augmented program satisfies them as well (properties are specified in branching-time logic CTL). A key assumption of their method is that after the aspect machine completes, the continuation is always to the state following the joinpoint in the original base program. This requirement is much stronger than the assumption used here of a weakly invasive aspect.

In [8], model checking tasks are automatically generated for the augmented system that results from each weaving of an aspect. That approach has the disadvantage of having to treat the augmented system, but at least the needed annotations and set-up need only be prepared once. That work takes advantage of the Bandera [5] system that generates input to model checking tools directly from Java code, and can be extended to, for example, the aspect-oriented AspectJ language. Bandera and other systems like Java Pathfinder [6] that generate state machine representations from code can be used to connect common high-level aspect languages to the state machines used in the results here.

In [7] a semantic model based on state machines is given, and the treatment of code-level aspects and joinpoints defined in terms of transitions, as in AspectJ, is described. In particular, the variations needed to express in a state machine weaving the meaning of *before*, *after*, and *around* with *proceed* are outlined, although work remains to fully capture the intended semantics.

In [4] and [11], among others, an assume-guarantee structure for aspect specification is suggested, similar to the specifications here, but model checking is not used.

6. CONCLUSION

By reusing the notion of a tableau which contains all possible behaviors that satisfy a particular formula, we can achieve a modular verification for aspects by augmenting the tableau with the advice according to a pointcut descriptor and ex-

amining the result. In order to do so we must restrict our view to aspects which are weakly invasive and always return to states which were reachable in the original base system, and we take a liberal view of fairness in which any computation that infinitely often visits an aspect state is considered fair.

A number of directions for future work present themselves quite clearly. While the current technique only addresses a single aspect and pointcut descriptor, in principle it can be extended to work for multiple aspects, given proper definitions of the weaving mechanics. Further development of how weaving is formulated will also allow treatment of cases where advice introduction changes the set of joinpoints. Furthermore, the entire discussion here is given in terms of states and state machines, while, as noted earlier, the usual basic vocabulary of aspect-oriented programming talks about events. The language-level aspect terminology and problems of real object systems still must be fully expressed in the state-based model checking used here. Nevertheless, the generic method in this paper allows us for the first time to model check aspects independently of a concrete base program, and is a significant step toward the truly modular verification of aspects.

7. REFERENCES

[1] E. Abraham, F. de Boer, W.-P. de Roever, and M. Steffen. An assertion-based proof system for multithreaded java. *Theoretical Computer Science*, 331(2-3):251–290, 2005.

[2] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Y. Vardi. Regular vacuity. In D. Borrione and W. Paul, editors, *Proc. of Correct Hardware Design and Verification Methods, CHARME'05*, volume 3725 of *LNCS*, pages 191–206. Springer, 2005.

[3] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[4] B. Devereux. Compositional reasoning about aspects using alternating-time logic. In *Proc. of Foundations of Aspect Languages Workshop (FOAL03)*, 2003.

[5] J. Hatcliff and M. Dwyer. Using the Bandera Tool Set to model-check properties of concurrent Java software. In K. G. Larsen and M. Nielsen, editors, *Proc. 12th Int. Conf. on Concurrency Theory, CONCUR'01*, volume 2154 of *LNCS*, pages 39–58. Springer-Verlag, 2001.

[6] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), Apr 2000.

[7] S. Katz. Aspect categories and classes of temporal properties. In *Transactions on Aspect Oriented Software Development, Volume 1, LNCS 3880*, pages 106–134, 2006.

[8] S. Katz and M. Sihman. Aspect validation using model checking. In *Proc. of International Symposium on Verification*, LNCS 2772, pages 389–411, 2003.

[9] S. Krishnamurthi, K. Fisler, and M. Greenberg. Verifying aspect advice modularly. In *Proc. SIGSOFT Conference on Foundations of Software Engineering, FSE'04*, pages 137–146. ACM, 2004.

[10] NuSMV. `http://nusmv.irst.itc.it/`.

[11] H. Sipma. A formal model for cross-cutting modular transition systems. In *Proc. of Foundations of Aspect Languages Workshop (FOAL03)*, 2003.