Modular Verification of Strongly Invasive Aspects

Emilia Katz Shmuel Katz
Computer Science Department
Technion – Israel Institute of Technology
{emika, katz}@cs.technion.ac.il

ABSTRACT

An extended specification for aspects, and a new verification method based on model checking are used to establish the correctness of strongly-invasive aspects, independently of any particular base program to which they may be woven. Such aspects can change the underlying base program variables to new states, and after the aspect advice has completed, the base program code continues from states that were previously unreachable. The needed changes in the MAVEN model checker are described, and the soundness of the verification method is proven. An example is shown of its application to aspects that provide various bonus points to student grading programs.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model Checking*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms

Verification, languages

Keywords

Aspects, model-checking, specification, modularity

1. INTRODUCTION

Several works have dealt with model checking of aspect systems [8, 12, 5, 4, 9, 6]. These works either treat a system with aspects woven in, or try to deal with the aspects modularly, relative to a specification. In the later case, the motivation is either to reduce the size of the models, or to allow convenient reuse of aspects in a library. Such an approach requires that the aspect itself have an independent specification that can be shown to hold. In one form or another, the specification of an aspect describes an *assumption* about any base system to which the aspect can be woven, and a *guarantee* about the resultant system after the aspect is woven. The aspects are shown correct relative to their specification, and not to interfere with each other [6], and then, for each system to be constructed with the aspects, the base system is shown to satisfy the assumptions of the needed aspects. The construction of a model of the entire concrete woven system (which might be considerably larger than either of those used in the modular verification) and its direct verification do not have to be carried out at all.

So far, when aspects are treated separately from a specific weaving, it has been necessary to add a restriction: that the aspect returns control to the base system in a state that already existed for some computation of the base system without the aspect woven into it. Such aspects are called *weakly invasive* in [7], where the other categories of aspects mentioned in this paper are also defined. The reasoning behind the restriction is easy to understand: the aspect's assumption about the base system only relates to those computation sequences and states (known as *reachable* states) that can occur for some fair execution of the base system without the aspect. When an aspect returns control to the base system code, but in a state of the base variables that does not occur for any computation of the base system that begins from a "normal" initial state, there is no restriction on the behavior of the continuation. Instructions from the base code are executed, but with values that were never expected or tested, and with no restriction on the outcome. Thus the overall behavior of such a system is hard to analyze in a modular manner, separating the reasoning about the base from the reasoning about the aspects to be woven. In such cases, modular reasoning was thought unfeasible.

On the one hand, this restriction still allowed treating most aspects. Several kinds of aspects, including *spectative* ones that merely gather information, and *regulative* ones that merely restrict possible steps, are weakly invasive. Moreover, often the category of such aspects can be identified using dataflow techniques, as described in [7, 11, 13], and many commonly used aspect examples are weakly invasive. Nevertheless, there are other aspects that definitely are strongly invasive, and that occur in real applications, so that a more complete approach is desirable.

In this paper we show that such a restriction is unnecessary, and that a modular approach can be realized even for so-called *strongly invasive* aspects that do return control to the base system in new states that were unreachable in the base system executing alone. To do this, we take advantage of the usual organization of model checkers for linear

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2009 All rights are reserved by the authors

time systems, and of the facilities they commonly provide. An extension of the MAVEN aspect verification system is presented, that can treat strongly invasive aspects, and an example of a bonus aspect for student grades is described.

The basic idea of the new approach is to add to the specification an assumption about the base system that restricts the computation segments that may become reachable after a strongly invasive aspect is woven. We then show once-and-for-all that when the aspect is woven into any base system with a reachable part that satisfies the previous type of assumption and an unreachable part that satisfies the added one, the result of the weaving will satisfy the guarantee. For a particular base system, we then have to show that the assumptions are true for both the reachable and unreachable parts (or at least the unreachable part that may become reachable after weaving). These tasks are made feasible due to the fact that many model checkers actually generate a state transition system that includes the unreachable parts of the computation, as a side-effect of the construction, and that marking the reachable states is a built-in operation.

The original MAVEN system [4], over NuSMV [1], builds a single model that can be checked to establish the correctness of a weakly-invasive aspect relative to its assume-guarantee specification, given in Linear Temporal Logic (LTL). (In the examples in this paper we use only the LTL modalities $\mathsf{G}\,p$ - for "from now on, p", $\mathsf{F}\,p$ - for "eventually, p", and $p\ \mathsf{U}\ q$ - for "p is true until q becomes true"). The tableau state machine of the assumption is built using a module of NuSMV, and then the transition system of the aspect advice is woven into it, with pointcuts defining transitions to the beginning of advice state machine fragments, and with transitions back to the states of the base system that match the end states of the advice segments. It is then proven that whenever this particular model satisfies the guarantee assertion, then a woven system with any base satisfying the assumption, and the model corresponding to the aspect woven into it, satisfies the guarantee.

In the following section, precise definitions of the terms involved are presented, the theory behind the verification algorithm is described, and a proof of soundness is given, that extends the one given for the simpler MAVEN system. In Section 3 algorithms are given for computing the last states of the aspect, for determining the category of the aspect, for verifying the aspect, and for checking the base system for the needed assumptions. In Section 4 the specification and verification of an aspect for adding bonus points for student exercises and exams is described, and some concluding remarks are in Section 5.

2. VERIFICATION THEORY FOR STRONGLY INVASIVE ASPECTS

DEFINITION 1. *An aspect A is* strongly invasive *relative to a model M if a state of M that was unreachable in M becomes reachable in the woven system M+A and transitions of M are applied to it.*

The last part of the definition is needed to ensure that the aspect advice (sometimes) finishes in a state of M that was previously unreachable, and then the code of M is applied to the new state.

2.1 Refined Aspect Specification

The assumption of a strongly invasive aspect has to contain more information than the assumption of a weakly invasive one: it sometimes needs to define restrictions on the behavior of the unreachable part of the base system into which the aspect can be woven, in order to ensure an appropriate behavior of the woven system from the states that are made reachable by the strongly invasive aspect.

The specification of aspect A is now a triple: (P_A, U_A, R_A), where, as before (in [4]), P_A is the assumption about the reachable part of the base system and R_A is the result assertion guaranteed to hold in the woven base with the aspect. The new U_A statement is an LTL formula defining the restrictions on the unreachable part of the base system which is made reachable by completing an aspect advice fragment. The restriction is posed on computations of the base system that start in the states that might be reached by completing the aspect advice, which were previously unreachable. We now may define the correctness of an aspect relative to such a specification, relating to a base system $S = S_{reach} \cup S_{unreach}$ where S_{reach} represents the reachable part, and $S_{unreach}$ the unreachable part.

DEFINITION 2. *An aspect A is* correct *with respect to its refined assume-guarantee specification (P_A, U_A, R_A) if, whenever it is woven (by itself) into a system $S = S_{reach} \cup S_{unreach}$, where S_{reach} satisfies P_A and the part of $S_{unreach}$ that might become reachable after weaving satisfies U_A, the result will satisfy the guarantee, R_A.*

The property of the unreachable part of the system is relevant only for computation segments starting from a state that can be the last state of an advice execution. The reason is that only by an advice execution can a computation of the woven system pass from a state that was reachable in the base system to a state which was unreachable in the base system. Thus in order to check that the unreachable part of the base system satisfies the requirements of the aspect, it is enough to verify a formula of the form $L_A \rightarrow U_A$ on it, where L_A is a state formula describing the set of all the possible last states of the advice state machine, projected on the base system variables.

With some abuse of notation, we denote by L_A the set of possible last states of aspect A (identifying the unary predicate with the set it describes). Note that this set consists exactly of all the states in the base system into which a computation can arrive after finishing advice execution.

2.2 Refined Tableau Construction

Given an aspect A and its refined specification, (P_A, U_A, R_A), we need to construct a refined tableau to serve as a representation of all the base systems into which our aspect will possibly be woven. But now in order to build the tableau of the assumption of the aspect, it is not enough to build the tableau of P_A: we need to restrict the unreachable part of the tableau. The tableau needs to represent the systems, the reachable part of which satisfies P_A, and the unreachable part of which satisfies $L_A \rightarrow U_A$, where L_A is the predicate defining the set of all the possible return states of the advice. The refined tableau, T, is constructed in three steps:

Step 1: Automatically construct the predicate L_A. The construction is shown in Section 3.1.

Step 2: Use the `ltl2smv` module of the NuSMV model checker to build the tableau T_1 of the LTL formula $(P_A \vee (L_A \wedge U_A))$.

Step 3: Take the tableau T to be the same as T_1 except for the initial states definition. To obtain the $INIT$ predicate of T, restrict the $INIT$ predicate of T_1 to include only states that should be reachable in the base system: $INIT \wedge P_A$.

Note that T_1 is the tableau of $(P_A \vee (L_A \wedge U_A))$ and not of $(P_A \vee (L_A \rightarrow U_A))$, because the only way to reach the part of the base system that does not satisfy P_A is by application of an aspect advice, and this will bring the computation to a state in which L_A must hold. This intuition will be justified during the proof of Theorem 1.

Let us denote the refined tableau constructed as above by $T_{(P_A,(U_A,L_A))}$.

THEOREM 1. *Let A be an aspect with the refined assume-guarantee specification (P_A, U_A, R_A), and let L_A be a formula describing the set of all the possible last states of A. Then A is correct with respect to (P_A, U_A, R_A) if the result of weaving A into $T_{(P_A,(U_A,L_A))}$ satisfies R_A.*

We delay the proof of the theorem until after bringing some helpful definitions and lemmas needed for the proof. They appear below, together with the intuition for the proof.

In order to prove the theorem we need to show that if the result of weaving A into $T_{(P_A,(U_A,L_A))}$ satisfies R_A, then for every base system M such that its reachable part satisfies P_A and the unreachable part satisfies $L_A \rightarrow U_A$, the result of weaving A into M satisfies R_A. For this purpose it is enough to show that for every infinite fair path σ in the woven system $M + A$ there exists a corresponding infinite fair path π in the woven tableau, $T_{(P_A,(U_A,L_A))} + A$, such that $label(\sigma) \mid_{AP} = label(\pi) \mid_{AP}$. (Where AP is the set of all the atomic propositions appearing in the specification of A, a label of a state s, $label(s)$, is the set of all the atomic predicates that hold at the state s, and a label of a path τ, $label(\tau)$, is defined to be the sequence of the labels of the states of τ, so that if $\tau = s_0, s_1, s_2, \ldots$, $label(\tau) = (label(s_0), label(s_1), label(s_2), \ldots)$). In that case indeed in order to prove that every path in the woven system satisfies R_A, it is enough to show that every path in the woven tableau satisfies this property.

To simplify the notation, let us denote $T_{(P_A,(U_A,L_A))}$ by T. The task of finding a fair path in $T + A$ that corresponds to the given fair path of $M + A$ will be divided into steps according to prefixes of σ, and at each step a longer prefix will be treated. The following lemma will help to extend the treated prefixes:

LEMMA 1. *Let S be a system, and let $s_0, \ldots, s_k$ be states in S such that s_0 and s_k are reachable by a fair path from some initial state of S (the paths and the initial states for s_0 and s_k might be different), and for each $0 \leq j < k$, the transition (s_j, s_{j+1}) exists in S. Then there exists a fair computation in S which contains the sequence of states $s_0, \ldots, s_k$.*

Proof.
A computation is *fair* if it visits states from the *Fairness set* of the system model infinitely often. Let π_0 and π_k be fair computations in S in which s_0 and s_k occur, respectively. Then $\pi_0 = \sigma_0 \cdot s_0 \cdot \ldots$, and
$\pi_k = \ldots \cdot s_k \cdot \sigma_k$ for some σ_0 and σ_k. Let us take
$\pi = \sigma_0 \cdot s_0 \cdot s_1 \cdot \ldots \cdot s_k \cdot \sigma_k$. This is obviously a path in S, and it starts from an initial state, as did σ_0. Moreover, π is a fair computation, because it has the same infinite suffix, σ_k, as the fair computation π_k.
Q. E. D.(Lemma 1)

The following definition will be useful for identifying the "interesting" prefixes of the path σ:

DEFINITION 3. *Any infinite path π in a transition system can be represented as a sequence of* path segments - $\pi = \pi^0 \cdot \pi^1 \cdot \ldots$, *where each path segment π^i is a sequence of states such that:*

- *If $i = 0$, the first state of π^i is the initial state of π*
- *If $i > 0$, the first state of π^i is either an initial state of an advice or a resumption state of the base system (i.e., a state in the base system into which the computation arrives after an advice execution is finished)*
- *The last state of π^i is either a pointcut state or a last state of an advice (after which the computation returns to the base system), or the last state of the path, if π is finite*
- *There are no pointcut states and no last states of advice inside π^i (i.e., in the states of π that are not the first or the last state)*
- *π is the concatenation of the path segments of π in the order of their indices*

Note that the decomposition of a path to path segments is unique, and that, because of loops, there can be resumption states within a segment. Note also that we could have an infinite (last) segment - in the reachable part of the base, or in the unreachable part, or even in the aspect. In our case all the paths in question are infinite, so the last state of each finite path segment will be either a pointcut or a last state of an advice. A resumption state might be unreachable in the system before weaving - in case of a strongly invasive aspect.

Now if we are given a path of M+A, $\sigma = \sigma^0 \cdot \sigma^1 \cdot \ldots$ where σ^i-s are the path segments of σ, for each finite prefix of σ consisting of a number of path segments we define the set of *corresponding path-segment prefixes* of fair paths in T+A:

$$\begin{aligned} \Pi_i = \{&\pi^0 \cdot \pi^1 \cdot \ldots \cdot \pi^i \mid \\ &label(\pi^0 \cdot \ldots \cdot \pi^i) \mid_{AP} = label(\sigma^0 \cdot \ldots \cdot \sigma^i) \mid_{AP}, \\ &\exists \pi \textit{ fair path in T+A such that } \pi = \pi^0 \cdot \ldots \cdot \pi^i, \ldots\} \end{aligned}$$

Each element in Π_i is a prefix of an infinite fair computation of $T + A$ corresponding to the i-th prefix of σ, thus the following lemma will show that for every finite prefix of σ there exists a corresponding prefix of a fair computation in $T + A$:

LEMMA 2. *Given a fair computation σ of M+A, and sets of prefixes Π_i-s as defined above, $\forall i \geq 0. \Pi_i \neq \emptyset$.*

Proof.
The proof is by induction on i.

Base: $i = 0$.

To show that Π_0 is not empty we need to show the existence of π^0 such that $label(\pi^0) \mid_{AP} = label(\sigma^0) \mid_{AP}$ and π^0

is a prefix of some fair path π in T+A. σ^0 is the first path-segment of a fair path in $M+A$, thus there is no advice application before σ^0 or inside it. So σ^0 is also the first path segment of a fair computation in M. According to the assumption on M, $M \models P_A$, thus for every fair path starting from an initial state of M there exists a corresponding fair path in T. In particular, there exists a fair path $\pi = t_0, \ldots, t_k, \ldots$ in T such that *label*$(\sigma^0) \mid_{AP} =$ *label*$(t_0, \ldots, t_k) \mid_{AP}$. Then again, as $t_0, \ldots, t_k$ is a beginning of a fair path in T, and there are no pointcuts in it, except maybe for the last state, it is also a beginning of a fair computation in $T + A$. So let us take $\pi^0 = s_0, \ldots, s_k$. We are left to show that π^0 is indeed a path-segment, and then it will follow that $\pi^0 \in \Pi_0$, meaning that Π_0 is not empty.

label$(t_0) =$ *label*$((\sigma^0)_0)$, thus t_0 is an initial state of $T+A$. There is no pointcut inside σ^0, because it is a path-segment, so the last state of σ^0 cannot be a return state of advice application, which means that it has to be a pointcut state. Due to the agreement on labels, the last state of π^0 will also be marked as a pointcut state. For the same reason, there are no pointcut states among $t_0, \ldots, t_{k-1}$, which, in the same way as for σ^0, implies that there are no advice return states also. Thus both ends of π^0 are legal ends of a path-segment, and there are no pointcut states and no advice return states inside π^0, which makes it, indeed, a legal path-segment.

Induction step.

Let us assume that for every $0 \leq i < k$, $\Pi_i \neq \emptyset$. We need to prove that $\Pi_k \neq \emptyset$.

The induction hypothesis holds, in particular, for $i = k-1$, thus there exists some prefix $\pi^0 \cdot \pi^1 \cdot \ldots \cdot \pi^{k-1}$ of a fair computation of $T + A$, corresponding to the prefix $\sigma^0 \cdot \sigma^1 \cdot \ldots \cdot \sigma^{k-1}$ of $M + A$'s computation, σ. Let us denote by *s_first(i)* the first, and by *s_last(i)* the last state of $i - th$ path-segment of σ (σ^i), and symmetrically for the states of path segments of $T + A$ - by *t_first(i)* the first, and by *t_last(i)* the last state of $i - th$ path-segment. There are two possibilities for *s_last(k-1)*:

1. *s_last(k-1)* is a pointcut. Then *t_last(k-1)* is also a pointcut, because due to the induction hypothesis *label*(*s_last(k-1)*) $\mid_{AP} =$ *label*(*t_last(k-1)*) $\mid_{AP}$. Then in every continuation of the computation both in $M + A$ and in $T + A$ the advice of the aspect will be performed, thus the k-th path-segment will in both cases be the application of the same advice from the same state, and the agreement on the labels of the k-th path-segments will be trivially achieved. Moreover, for the same reason the existence of an infinite fair path with the prefix $\pi^0 \cdot \pi^1 \cdot \ldots \cdot \pi^{k-1}$ implies the existence of an infinite fair path with the prefix $\pi^0 \cdot \pi^1 \cdot \ldots \cdot \pi^k$, because every continuation of the first prefix had to be an advice application. From the above it follows that in this case $\Pi_k \neq \emptyset$.

2. *s_last(k-1)* is a last state of the advice. This, in particular, implies that *s_last(k)* is a pointcut state, and no advice has been applied between *s_last(k-1)* and *s_last(k)*. Here are again two possibilities:

   - *s_last(k-1)* is a reachable state in M (more precisely, the state reachable in M is the projection of *s_last(k-1)* on AP). As no advice is applied between *s_last(k-1)* and *s_last(k)*, we have that the whole path-segment σ^k is in the reachable part of M. Moreover, due to Lemma 1, as both *s_last(k-1)* and *s_last(k)* are reachable by some fair paths from some initial states of M, we also have that there exists a fair computation of M containing the sequence *s_last(k-1)*, *s_first(k)*, $\ldots$, *s_last(k)*. All the fair computations of the reachable part of M are represented in the tableau of P_A, which is exactly the reachable part of T. Thus, in particular, the above fair path has a corresponding path in T, and, as there was no pointcut or advice application inside the sequence *s_last(k-1)*, *s_first(k)*, $\ldots$, *s_last(k)*, there are also no pointcuts and advice applications in the corresponding sequence in the computation of T, and thus there exists a corresponding sequence of states in $T+A$, π^k. The first state of π^k, *t_last(k-1)*, is reachable from the initial state of $T + A$ by some fair path, as Π_{k-1} is not empty. Moreover, all the prefixes of such fair pathes appear in Π_{k-1}, thus at least one of them continues to the sequence π^k. So indeed we obtain that there exists a sequence of states π^k corresponding to σ^k in the woven tableau, for which a fair continuation exists. We are left to see that the sequence of states, π^k, is indeed a path segment in the woven tableau computation. But this is true due to the agreement on labels of the states, *label*$(\pi^k) \mid_{AP} =$ *label*$(\sigma^k) \mid_{AP}$: the path segment σ^k started from a return state of an advice, ended by a pointcut, and had no advice applications in the internal states, so the same is true for π^k and thus π^k is a path segment.
   - The last case left is that *s_last(k-1)* is unreachable in M. Additionally, *s_last(k-1)* is the last state of the advice, thus *s_first(k)* is the return state of the advice, and also is unreachable in M, because according to the weaving algorithm *label*(*s_last(k-1)*) $\mid_{AP} =$ *label*(*s_first(k)*) $\mid_{AP}$. From the fact that *s_first(k)* is unreachable in M, together with the assumption on the unreachable part of M, we have that $L_A \rightarrow U_A$ holds in the suffix of any path starting from *s_first(k)*. But from the agreement on labels with *s_last(k-1)* we also have that *s_first(k)* $\models L_A$. Together we obtain that U_A holds in the suffix of any computation in M starting from *s_first(k)*, and, in particular, for the computation $\sigma\prime$ containing the next path segment of σ, σ^k (because there is no advice application inside σ^k, all its states are states of the original system, M - either in the reachable or the unreachable part). Now let us examine the states of the woven tableau. The tableau of $L_A \wedge U_A$ is included in the refined tableau T, thus every computation satisfying U_A that starts from a state satisfying L_A is represented in T (though its initial state might be unreachable before the aspect is woven into T). Let $\pi\prime$ be a computation that corresponds to the suffix of $\sigma\prime$ that starts from *s_first(k)*. The first state of $\pi\prime$ agrees on its label with *s_first(k)*, and thus with *s_last(k-1)*, which, according to the induction hypothesis, implies agreement on labels with *t_last(k-1)*. According to the weaving algorithm, the last state

of the advice is connected to all the states in the underlying system with which it agrees on labels. Thus, in particular, *t_last(k-1)* (which is the last state of the advice, in the same way as *s_last(k-1)*), is connected to the first state of $\pi\prime$. So we can take the first state of $\pi\prime$ to be the first state of π^k. Let us then take π^k to be the first path-segment of $\pi\prime$. It is indeed a path segment of a fair computation (due to Lemma 1), it is connected to π^{k-1} and agrees on labels with σ^k, so we found what we needed.

Thus, indeed, the set of possible continuations, Π_i, is never empty.
Q. E. D.(Lemma 2)

Theorem 1 proof:.
Now let us return to the proof of Theorem 1. Let us be given an infinite fair path σ in the woven system $M + A$. From Lemma 2 it follows that there exists an infinite path π in the woven tableau corresponding to the given path σ - all the prefixes of π appear in the Π_i-s above, and due to the lemma, the Π_i-s are all non-empty. So in order to complete the proof of the theorem we need only to notice that every path constructed from the prefixes in Π_i-s above is fair, for the following reason: There are two possibilities for the infinite suffix of π. It either has infinitely many advice applications, or there exists some infinite suffix in which no aspect state is visited. If there are infinitely many advice applications, some state of the advice must be visited infinitely often, and all the states of the advice are defined as fair. If there is no advice application after some state, then there are only a finite number of path segments of π, and the last path segment is infinite. But, as we know, this path segment belongs to some fair path in $T + A$, so this must be a fair suffix, and so the computation π is indeed fair. This completes the proof of Theorem 1
Q. E. D.

3. ALGORITHMS

3.1 Computing L_A Automatically

Given a model of the aspect, A, in MAVEN format, we would like to automatically compute the state formula defining the set of all the possible last states of A's advice. The algorithm we propose consists of four steps:

Step 1: Construct a formula φ defining the pointcut of the aspect: take φ to be the disjunction of all the POINTCUT expressions in A.

Step 2: Run MAVEN on a model $A\prime$ which is the same as A except for a change in the specification. The assumption of the aspect is replaced by φ, and the guarantee of the aspect is replaced by *true*. The purpose of this operation is to obtain a system in which all the possible computations of the aspect are represented, and this goal is achieved in the following way:

- At the first step of its work, MAVEN will automatically construct the tableau of the new assumption of the aspect, φ, using the `ltl2smv` module of NuSMV. Note that in this tableau, T_φ, only the initial states are restricted, and the initial states are exactly all the possible join-points of the aspect.
- At the second step, MAVEN will perform the weaving of the aspect into the constructed tableau. The obtained woven system, $T_\varphi + A$, will contain all the possible computations of the aspect, because the initial states of the tableau are all the possible pointcut states that can occur in either reachable or unreachable parts of the base systems into which A will be woven (as the ranges of all the base variables as defined in the aspect model definition are the maximal possible, and the combinations of variables values are restricted only by the formula φ).

Note that if we added other restrictions on the computations of the tableau T_φ, we may not be able to guarantee that all the possible runs of the advice of A will appear in the woven tableau. For example, if we demand that the computations of the tableau should satisfy P_A, then after the weaving we would not obtain the runs of the aspect from the states that were unreachable in the base system. Since in the unreachable part of the base system which becomes reachable after the weaving there might be join-points of A, we have to model the computations of the advice starting from these states. However, there are cases when additional restrictions might be posed on the computations of the tableau built. For example, there might be some invariant that holds both in the reachable and the unreachable parts of the base system, and then it could be added to φ. Additionally, there might exist an assertion that holds for all the pointcut states, but is not explicitly written as part of the pointcut. Then it would be possible to restrict the initial states of the constructed tableau by this assertion.

Step 3: Take the woven system obtained in Step 2, $T_\varphi + A$, and use the built in functionality of NuSMV to compute the set of all the reachable states of this model, $(T_\varphi + A)_{reachable}$. For each of the states in $(T_\varphi + A)_{reachable}$, check whether it satisfies any of the RETURN conditions of the aspect. If it does, add it to the set L_A.

Step 4: Now L_A is the set of all the possible last states of A. What is left is only to construct the predicate describing this set. This is done by taking the disjunction of all the predicates describing the states in L_A.

Sometimes it might be easy to see a compact description of the possible last states of the aspect. For this case we provide the user a possibility to supply a manually constructed predicate L. But such a predicate should be checked before use, because the intuition of the user might be wrong. Then we use the above algorithm to construct the full L_A predicate, and check that the supplied predicate L is implied by L_A. If indeed $L_A \rightarrow L$ holds, the verification using L will still be sound, because it just might check additional paths, but no relevant path will be left unverified.

3.2 Determining the Aspect Category

Before applying the full verification technique it is very desirable to determine the category of the aspect. If the aspect is of the weakly invasive category (or a simpler category included within the weakly invasive one), then the method described in [4] is applicable to it. Otherwise, the method described in Section 3.3 should be used.

Some ways of determining the category of the aspect using code analysis, dataflow techniques and semantic definitions are described in [7, 11, 13, 3]. If none of them gives a positive answer, the algorithm presented below can help to determine whether the aspect is uniformly strongly invasive, i.e., is always strongly invasive for every possible base system to which it can be woven. But first some definitions and observations are needed:

REMARK 2. *From Definition 1 in Section 2 it immediately follows that for any system M in which all the states not reachable from the initial state by some fair path have been removed, if an aspect A is strongly invasive relative to M, there is a deadlock in the system $M + A$: Let s be a last state of advice execution such that there exists no reachable state $s\prime$ in M for which label($s\prime$) = label(s) $|_{AP}$. Then this state is a deadlock state in the woven system.*

LEMMA 3. *Let aspect A have the specification (P_A, U_A, R_A), where AP is the set of all the atomic propositions appearing in the specification and T_P denotes the tableau of P_A. Aspect A is* strongly invasive *with respect to P_A if when A is woven into T_P, there exists a state s in $T_P + A$ such that:*

- *s is the last state of advice execution, and*
- *there exists no state $s\prime$ in T_P such that $s\prime$ is reachable by some computation of T_P and label($s\prime$) = label(s) $|_{AP}$*

Proof.
Immediate from the above remark.

DEFINITION 4. *Given a tableau T of an LTL formula ϕ, the tableau TP obtained from T by removing all the states that are not reachable from the initial state of T by any fair path (and only them) is called the* pruned tableau *of ϕ.*

Note that the above defined pruned tableau is equivalent to a tableau obtained from T by removing all the states and transitions that only lead to deadlock states.

LEMMA 4. *Aspect A with the specification (P_A, R_A) is strongly invasive relative to P_A iff there exists a deadlock in the system $TP_A + A$, where TP_A is the pruned tableau of P_A.*

Proof.
The conditions of Remark 2 above hold, in particular, for $M = TP_A$, so there will be a deadlock state in $TP_A + A$.

On the other hand, if there exists a deadlock in the system $TP_A + A$, let s be the deadlock state. Let us denote by $s\prime$ the state of TP_A such that *label*($s\prime$) = *label*(s) $|_{AP}$. There are two possibilities: If $s\prime$ is reachable in TP_A, then there exists some infinite computation $\pi = s\prime, s_2, \ldots$ from $s\prime$ in TP_A, because TP_A is a pruned tableau. In particular, there exists a state s_2 in TP_A (the second state of π) to which $s\prime$ is connected. However, in $TP_A + A$ the state s is no longer connected to s_2. According to the construction of $TP_A + A$, the only reason could be that an advice is applied at s. But if an advice was applied at s, s would not be a deadlock state. Thus when we assumed that the projection of s on AP is reachable in TP_A we obtained a contradiction. So we conclude that $s\prime$ is unreachable in TP_A.

But could $s\prime$ still be reachable in T_P? This can only be if $s\prime$ has been removed from T_P during the construction of the pruned tableau. This means that all the paths starting from $s\prime$ led to some deadlock states, and thus $s\prime$ couldn't be reached by any fair computation of T. But according to Lemma 3 this exactly means that the aspect A is strongly invasive relative to its assumption.
Q. E. D.

According to Lemma 4, the following algorithm verifies whether the given aspect is strongly invasive relative to its assumption:

1. Construct the pruned tableau TP_A from the tableau of the assumption of A. This is done automatically, by an iterative procedure that we have added to MAVEN. The procedure is as follows:
   - Run NuSMV to detect deadlock states in the tableau.
   - If a deadlock state is detected, construct a predicate describing this state, p
   - Rule out the deadlock state: Add the negation of p to the initial state definition, and to the predicate defining possible next states of the transitions.

   Repeat the procedure until there are no more deadlocks in the tableau.
2. Use MAVEN to weave the aspect into the above constructed tableau.
3. Run NuSMV to check whether there are deadlocks in the woven tableau. If a deadlock is detected, the aspect is strongly invasive relative to its assumption. Otherwise, the aspect A is weakly invasive relative to P_A.

Note that the algorithm presented here gives a positive answer only if the aspect is strongly invasive *relative to the tableau of its assumption*, but not relative to a concrete base system. Thus if the algorithm gives a positive answer, the aspect is strongly invasive relative to all the possible base systems into which it might be woven. But if the algorithm gives a negative answer, there might exist a base system satisfying the assumption of the aspect, with respect to which our aspect is still strongly invasive.

Given a base system S, there is one more way for us to check whether the given aspect, A, is strongly invasive relative to this system. Intuitively, what we would like to do is to look at all the unreachable states of the base system, and check whether there are last states of our aspect among these unreachable states. For that purpose we can check satisfiability of the following formula: $\varphi = S_U \wedge L_A$, where S_U is the formula defining the set of all the unreachable states of S, and L_A is the formula defining the set of all the possible last states of A. φ can be constructed automatically: the way to construct L_A automatically is shown in Section 3.1, and the way to construct S_U automatically is shown in Section 3.4.1. And then the satisfiability of φ can be automatically checked using a SAT solver (such as, for example, Chaff [10]). If φ is found unsatisfiable, it means that there are no last states of the aspect A in the unreachable part of S, so A has to be weakly invasive relative to S, and the simpler model check in [4] can be used. If φ is found satisfiable, it doesn't necessarily imply that A is strongly invasive relative to S, because the predicate L_A is an over-approximation: it contains all the possible last states of the aspect, but maybe some of them will never occur in the computations of the woven system $S+A$, and thus will not bring the computation to states that were unreachable in S. But this over-approximation is a safe one: if we declare some aspect as strongly invasive when it is weakly invasive, we will just have to work harder to prove its correctness than we would if we knew its exact category, but the verification results will be sound.

3.3 Verifying the Aspect

Given an aspect A and its refined assume-guarantee specification, (P_A, U_A, R_A), the verification of correctness of A with respect to (P_A, U_A, R_A) is performed as follows:

1. Construct the refined assumption tableau for A as shown in Section 2.2 - the $T_{(P_A,(U_A,L_A))}$.
2. Use MAVEN to weave A into $T_{(P_A,(U_A,L_A))}$ and to run the NuSMV model checker on the resulting system and check the R_A property on it.

3.4 Base System Correctness Verification

3.4.1 Non-optimized solution

Given a base system S, we need to verify that it satisfies the refined assumption of our aspect, (P_A, U_A):

- Verify that the reachable part of S, S_{reach}, satisfies P_A
- Verify that all the computations starting from the unreachable part of S, $S_{unreach}$, satisfy $L_A \rightarrow U_A$.

The first verification task can be done by usual model-checking of S versus P_A. The meaning of the second task is as follows: we need to examine the model of $S_{unreach}$ and check all the fair computations that start from states satisfying L_A (note that a computation starting from a state in $S_{unreach}$ might return to the reachable part of S at some state). All these computations should satisfy U_A. The verification is performed in three steps:

1. Automatically compute the state formula S_U defining the set of all the unreachable states of S: S_U is the negation of the formula S_R defining all the reachable states of S, and in NuSMV there exists a possibility to compute S_R automatically for a given system S.
2. In the model of the base system, S, automatically replace the initial states definition by the formula $S_U \wedge L_A$
3. Run NuSMV on the obtained model and the formula U_A. If the verification succeeds, it means that the given base system satisfies the restriction on the unreachable part.

3.4.2 Optimization

In some cases, the requirement in the second part of the verification process can be relaxed due to the structure of U_A. For example, in case when U_A is some safety property, i.e., U_A has the form $\mathsf{G}\,\varphi$, we do not have to verify that φ holds all along the computations starting from resumption states in the unreachable part of the system. We need to check only the segments between a resumption state and the next join-point or reachable state. So if we denote by *ptc* the predicate defining the pointcut of the aspect, and by *reachable* - the predicate defining the reachable states of the base system, then it is enough to verify the following formula on the unreachable part of the system: $L_A \rightarrow (\varphi\ \mathsf{U}\ (\mathit{reachable} \vee (\mathit{pointcut} \wedge \varphi)))$. The reason is that when the computation reaches a join-point, in the woven system the advice will be executed at that point, so the information about the possible continuations of the computation in the base system from that point is useless. And if a computation leaves the unreachable part and arrives to some previously reachable state, its continuation will behave as specified by the assumption of the aspect about the reachable part of the base system, and all these continuations are already checked during the reachable part verification.

As an example of the situation described above, we can take a look at an aspect that is in charge of the scheduling policy of a semaphore-guarded resource. The purpose of the aspect is to implement a possibility of a waiting queue for the semaphore. As a result, the semaphore that could previously have only values 0 or 1 can now have negative values (according to the number of waiting processes). Thus the aspect is indeed strongly invasive. But there is a part of the system invariant that we need to extend to the unreachable part of the base system: regardless of the semaphore value and the concrete scheduling algorithm, we demand that no two processes hold the guarded resource at the same time. So if the formula ψ encodes the fact that two processes hold the resource at the same time, the assumption of the aspect about the unreachable part of the base system should be $U = \mathsf{G}\neg\psi$. But when verifying the computations starting in the unreachable part of the base system, it is enough to check that after each possible last state of the aspect the computation satisfies $\neg\psi$ until it arrives to a pointcut state or to a reachable state.

4. EXAMPLE

In this example we discuss an aspect that can be used in any grades-managing system. The aspect B provides a way of giving bonus points for assignments and/or exams (thus making it possible to have assignment/exam grades that are more than 100), but still keeping the final grade within the 0..100 range.

The aspect has two kinds of pointcuts, and two corresponding pieces of advice. The first pointcut of B is the moment when an assignment or exam grade is entered to the system. At this point the original system would accept only grades between 0 and 100, but the aspect offers a possibility of giving a bonus on the grade, and stores the new grade successfully even if it exceeds 100. The second pointcut of B is the moment when the final grade calculation of the base system is performed. Then if the calculation resulted in a grade that exceeds 100, the aspect replaces this grade by 100 (otherwise keeping the grade unchanged).

Aspect B is strongly invasive in the systems into which it can reasonably be woven, because its operation results in states in which some grades are more than 100, which is impossible in the base systems without bonus policies. And this example, though simple, is still of interest to us, because the aspect here exhibits a typical behavior we would like to treat: when it is woven into a system, the calculations there are performed partly in the aspect, and partly in the base system code, but using new inputs, that were impossible before the aspect was woven in.

The specification of B can be formalized as follows:

- The assumption on the reachable part of the base system is that all the grades appearing in the grading system - homework assignment grades (*hw_i*), exam grades (*exam_j*), final grade (*f*) - are between 0 and

100, and after the final grade is ready (*f_ready*) (i.e., all the assignments and exams that comprise the grade have been checked, and the final grade has been calculated from them according to the base system grading policy), the final grade is published (*f_published*). The result of the final grade calculation is represented by *calc*.

$$\begin{aligned} P_B = [\, & \mathsf{G}(f\_ready \rightarrow ((f = calc) \wedge \mathsf{F}\, f\_published)) \\ & \mathsf{G}(f\_published \rightarrow f = calc) \wedge \\ & \mathsf{G}(0 \leq f \leq 100) \wedge \\ & \mathsf{G}(\forall 1 \leq \ i \leq 10 (0 \leq hw\_i \leq 100)) \wedge \\ & \mathsf{G}(\forall 1 \leq \ j \leq 2 (0 \leq exam\_j \leq 100))] \end{aligned}$$

Here, for modeling purposes, we have to provide some bounds on the number of assignments and exams, so we assume that there are no more than 10 home assignments and no more than 2 exams in each course. We also show the specification for the grades of a single student (because the grades of different students are independent, and calculations involving them can be viewed as orthogonal). When the model of the aspect is built, the ranges of all the variables - both the aspect variables and the relevant base system ones - are defined. Let us assume, for example, that our aspect gives bonuses in range of 0..20 points, then all the grade variables defined in the model of B are in the range 0..120.

- The assumption on the unreachable part of the base system is in our case a weakening of P_B. We still want the final grades to be published after they are ready, but now the final and the intermediate grades do not have to be bound by 100, but by 120. So we are left with the following property:

$$\begin{aligned} U_B = [\, & \mathsf{G}(f\_ready \rightarrow ((f = calc) \wedge \mathsf{F}\, f\_published)) \wedge \\ & \mathsf{G}(f\_published \rightarrow f = calc) \wedge \\ & \mathsf{G}(0 \leq f \leq 120) \wedge \\ & \mathsf{G}(\forall 1 \leq \ i \leq 10 (0 \leq hw\_i \leq 120) \wedge \\ & \mathsf{G}(\forall 1 \leq \ j \leq 2 (0 \leq exam\_j \leq 120))] \end{aligned}$$

- The guarantee of the aspect now is that regardless of the existence of bonuses on the components of the final grade, the final grade will be the one calculated by the base system function, but rounded down to 100 if needed:

$$R_B = [\, \mathsf{G}(f\_published \rightarrow f = min(calc, 100))]$$

The guarantee of the aspect might also include a statement about the bonus policy it enforces, saying that the aspect calculates the bonuses as desired. But to simplify the discussion, we omit it here.

- The pointcut of the aspect can be formalized using the following predicates, which define the moments when the grades are entered into the system: *enter_hw_i* for homework grades, and *enter_exam_j* for exam grades.

$$\begin{aligned} Pointcut_B = [(& \bigvee_{i=1}^{10} (enter\_hw\_i)) \vee \\ & (enter\_exam\_1) \vee (enter\_exam\_2) \vee \\ & (f\_ready \wedge (f > 100))] \end{aligned}$$

Let us follow the verification algorithm, applying it to aspect B. The first step is the refined tableau construction. It begins with calculating the predicate L_B, defining all the possible last states of B. In our example, we get

$$\begin{aligned} L_B = [& (f\_ready \rightarrow ((f = 100) \wedge (calc > 100))) \wedge \\ & (\neg f\_published) \wedge \\ & \forall 1 \leq \ i \leq 10 (\neg enter\_hw\_i) \wedge \\ & \forall 1 \leq \ j \leq 2 (\neg enter\_exam\_j) \wedge \\ & (0 \leq f \leq 120) \wedge (0 \leq calc \leq 120) \wedge \\ & \forall 1 \leq \ i \leq 10 (0 \leq hw\_i \leq 120) \wedge \\ & \forall 1 \leq \ j \leq 2 (0 \leq exam\_j \leq 120)] \end{aligned}$$

And here is the explanation: All the combinations of exams and assignments grades values in range 0..120 are possible at the last state of the aspect, because all the grades of assignments and exams are independent. There is a connection between the final grade and the other grades, but only when the final grade is declared to be ready and still is not published. Then the final grade is equal to the minimum between the calculated value (*calc*) and 100. However, as we do not want to restrict the calculation function of the base system, we cannot establish this connection, and at the other states of the computation the value of the final grade is not restricted (except by its range), so effectively we have to enable any combination of the final grade value and the other grades. The values of the other system variables are restricted as follows: The variables *enter_hw_i* and *enter_exam_j* for all i-s and j-s are $false$, because no grade is entered by the user at the last state of the advice. The variable *f_published* is also $false$, because the aspect does not publish the grades - even if it was called at the moment when the final grade was calculated, it just modifies the calculated grade, but does not publish it. Publishing the grades is done by the base system. The next variable to discuss is *f_ready*. If the aspect was called at the moment of grades entering, the variable *f_ready* is $false$ at the join-point. The final grade is not calculated by the aspect in this case, so the variable remains $false$ at the last state of the advice. However, if the aspect was called at a join-point when the final grade is calculated, the variable *f_ready* is true there and remains true after the advice finishes its execution. In this case, as we said earlier, we will also have $f = 100$ and $calc > 100$.

Now after the predicate L_B is constructed, the tableau of the $(P_B \vee (L_B \wedge U_B))$ formula is created, its initial states are restricted to those satisfying P_B (that is, the refined tableau $T_{(P_B,(U_B,L_B))}$ is built), and then B is woven into the result. The last part of the verification process is running NuSMV on the woven tableau in order to check the R_B property on it. And for the above described aspect, with the specification given, the verification succeeds, so our algorithm shows that indeed it is correct with respect to its refined assume-guarantee specification. Intuitively, the reason for the success of the verification is that the base system performs only some arithmetic operations on the grades the aspect modifies, and thus we can expect that the result of performing old operations on the new arguments will be as anticipated, if only there is no overflow or type declaration problem. (By a type declaration problem we mean, for example, the case when the type of the grades variables is defined in the base code by some *typedef* to be 0..100, so that larger values cause a fatal type error.) But the assertion U_B ensures that this will not happen, because U_B will not hold for the base system in case such problems arise.

Note that the aspect does not restrict the grade calculation process of the base system, so this aspect is highly reusable, as long as the calculation can handle values greater than 100 (as seen in U_B). Moreover, this aspect can appear in a library of aspects providing different grading policies: different types of bonuses for homework assignments, or factors on the exam grades. All these aspects will have the same requirements from the base system as B does, so when some grading system is checked for applicability of one of the aspects from this library, it is automatically inferred that all the other aspects from the library are also applicable to this base system. Thus the grading policy can be changed as needed at any time, by replacing the applied aspect, without any further checks on the base system.

5. CONCLUSIONS

We have shown that strongly invasive aspects can be specified and shown correct relative to their specification, independently of a particular base system. Moreover, it is reasonable to check the properties needed from the unreachable part of the base system because the possible transitions of the base are considered bottom up, independently of the initial states, thus generating the unreachable part of the base system as a byproduct of model checking. Strongly invasive aspects typically extend the functionality of the base system to situations not originally covered. The examples seen in the paper, of a semaphore with negative values, and of aspects to give bonus points beyond the normal range, are typical. Often some invariants true in the base system alone will no longer hold after weaving such aspects, but other invariants will continue to hold, and are essential to the correctness of the woven system.

The verification method presented here is modular, and thus has an advantage over a straightforward non-modular verification of a woven system: the possibility of reuse without proof. There are two types of such reuse we see, both of which are demonstrated by the aspect described in Section 4. One case is when one and the same aspect is applicable to different base systems. Then the verification of the advice versus the assume-guarantee specification is performed only once, and in order to be able to apply the aspect to a given base system we need only to perform the base system verification described in Section 3.4. Another case is when a library of aspects is given, where all the aspects are built for the same purpose (like defining some action policy) and have a common assumption (P, U) about the base system. Then if we have a base system that satisfies the above assumptions, we can change the policy defined in this system at any time, by applying different aspects from the library - one at a time, of course - without any further checks.

When model-checking is used, the size of the verified system and of the specification is very important, as it strongly affects the verification time, and sometimes, if the model verified is too large, the model-checker can even fail to provide any answer. For the complexity analysis purpose, we denote by m the size of the base system model, by a - the size of the aspect model ($|A|$), by r - the size of all the formulas in A's specification (assuming, without loss of generality, that all the formulas used in verification - P_A, U_A, R_A - are approximately of the same size). When a formula of size k is verified on a model of size m, the space complexity of the model checking is $O(m \cdot 2^k)$ ([2]). Thus the complexity of a straightforward verification of the woven system is $O(2^r \cdot (m \cdot a))$, because a system of size m is verified against a formula of size r (the guarantee of A, in this case). Let us find the complexity of the modular verification method. It is the sum of the following components:

- The verification of the base system. It is of $O(2^r \cdot m)$ for the reachable part, and the same for the unreachable part, so together we obtain $2 \cdot O(2^r \cdot m) = O(2^r \cdot m)$
- Verification of the aspect. Here, first the refined assumption tableau is constructed, $T_{(P_A,(U_A,L_A))}$. The complexity of this step is $O(2^{2r})$ (Note that L_A is always a state formula, and thus does not increase the complexity.) Then the woven tableau is built, and we obtain a system of size $O(2^{2r} \cdot a)$. At the last step, the woven tableau is verified against the guarantee of A, R_A, and this requires complexity of $O(2^r \cdot (2^{2r} \cdot a))$

The total complexity thus is $O(2^r \cdot (2^{2r} \cdot a)) + O(2^r \cdot m)$. But the size of the base system model is usually very large, so $m \geq 2^{2r}$, and thus the complexity of our verification is usually not worse than that of the straightforward woven system check. Even when this is not the case, the possibilities for reuse make the modular approach preferable.

6. REFERENCES

[1] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proc. Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in LNCS, pages 495–499. Springer, July 1999. NuSMV home page: `http://nusmv.itc.it`.

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

[3] S. Djoko Djoko, R. Douence, and P. Fradet. Aspects preserving properties. In *Proc. of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantic-Based Program Manipulation (PEPM'08)*, pages 135–145. ACM, 2008.

[4] M. Goldman and S. Katz. MAVEN: Modular aspect verification. In *Proc. of TACAS 2007*, volume 4424 of *LNCS*, pages 308–322, 2007.

[5] E. Katz and S. Katz. Verifying scenario-based aspect specifications. In *Proc. Formal Methods: International Symposium of Formal Methods Europe (FM'05)*, volume 3582 of *LNCS*, pages 432–447. Springer, 2005.

[6] E. Katz and S. Katz. Incremental analysis of interference among aspects. In *Proc. of the 7th workshop on Foundations of aspect-oriented languages FOAL '08*, pages 29–38. ACM, 2008.

[7] S. Katz. Aspect categories and classes of temporal properties. *Transactions on Aspect Oriented Software Development (TAOSD)*, 1:106–134, 2006. LNCS 3880.

[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 and K. Fisler. Foundations of incremental aspect model-checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(2), 2007.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of the 38th Design Automation Conference, DAC'01*, pages 530–535, 2001.

[11] M. Rinard, A. Salcianu, and S. Bugrara. A classification system and analysis for aspect-oriented programs. In *Proc. of International Conference on Foundations of Software Engineering (FSE04)*, 2004.

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

[13] N. Weston, F. Taiani, and A. Rashid. Interaction analysis for fault-tolerance in aspect-oriented programming. In *Proc. Workshop on Methods, Models, and Tools for Fault Tolerance, MeMoT'07*, pages 95–102, 2007.