Incremental Analysis of Interference Among Aspects

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

ABSTRACT

Often, insertion of several aspects into one system is desired and in that case the problem of interference among the different aspects might arise, even if each aspect individually woven is correct relative to its specification. In this type of interference, one aspect can prevent another from having the required effect on a woven system. Such interference is defined and specifications of aspects are described. An incremental proof strategy based on model checking pairs of aspects for a generic model expressing the specifications is defined. When an aspect is added to a library of noninterfering aspects, only its interaction with each of the aspects from the library needs to be checked. Such checks for each pair of aspects are proven sufficient to detect interference or establish interference freedom for any order of application of any collection of aspects in a library. Implemented examples of interfering aspects are analyzed and the results are described, showing the advantage of the incremental strategy over a direct proof in space needed for the model check. Early analysis and detection of such interference in libraries of aspects will enable informed choice of the aspects to be applied, and of the weaving order.

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, interference, model-checking, detection, specification

1. INTRODUCTION

Aspects have proven useful for a wide variety of tasks, and aspect languages, such as, e.g., AspectJ [12], have become increasingly popular. Often reuse of aspects is desired, as the same concern might arise in many different systems. In some cases there is more than one aspect we would like to reuse in multiple systems of similar purpose, and then a library of reusable aspects is created. This is the case, for example, in [13], where a library of aspects was created to implement the ACID properties for transactional objects.

However, use of aspects raises many questions of reliability and correctness. It is crucial to establish both that each aspect individually is correct when woven alone, and also to consider possible interference among multiple aspects woven to the same underlying system. This paper considers the second question, giving a precise definition of semantic interference among aspects, showing how to detect it, and how to use examples of interference to modify the aspects or their specifications. The technique is most appropriate for establishing usage guidelines for reusable aspects, especially as libraries of reusable aspects are developed.

We define an incremental proof strategy based on model-checking that establishes whether there exists a legal underlying system in which the aspects interfere. For this strategy, assume-guarantee specifications of aspects described in Section 2 are used to define interference freedom in a way analogous to interference freedom among processes in shared-memory systems [16]. In that classic work, interference freedom among processes is defined in terms of whether independent and local Hoare-logic proofs of correctness for each parallel process are invalidated by operations from other processes. The individual proofs that each aspect is correct when woven alone correspond to the n local proofs of [16], while the approach in this paper deals with the n^2 checks of interference-freedom. A key point, also adapted here, is that the other processes may change the values of shared variables, but there is no interference as long as the independent proofs are not invalidated. The level of interleaving in shared memory systems is much finer than for aspects: every local assertion about memory values can be invalidated by another assignment by a different processor. The fact that aspect advice is only activated at joinpoints means that less stringent conditions can be used, and that modular model checking can be used as a proof component.

The work presented here is the first definition of semantic interference for aspects that uses the specification of the aspects as the interference criterion, and applies model checking to detect interference or establish noninterference among collections of aspects. In our method the interference checks are performed on pairs of aspects, and the results of these pairwise checks are shown to be sufficient to determine interference freedom for all the aspects in the library. However, as shown in Section 4, to enable such incremental proofs we have to "pay" by additional incompleteness.

There has been previous work on detecting whether the pointcuts of aspects match common joinpoints or overlapping introductions [4, 7]. This is important because the semantics of weaving can be ambiguous at such points, and be the source of errors. However, as will be shown, aspects can interfere even if there are no common joinpoints.

Some work has also been done in identifying potential influence by using dataflow techniques showing that one aspect changes (or may change) the value of some field or variable that is used and potentially affects the computation done by the advice of another aspect [17]. Slicing techniques for aspects [20, 1, 18] can also be used for such detection. Since such potential influence is often harmless, many false positives can result.

Interferences between an aspect and the base program were discussed in [11]. The reasoning about the influence of the aspects on the base system is based on the assume-guarantee paradigm, which is also used in our specifications of aspects. However, in [11] interactions between aspects are not considered, and no automated verification procedure is presented.

Verification based on model checking has mainly concerned the verification of a single aspect relative to an underlying system. In [14] a modular approach is presented to show that assertions true of a given underlying system remain true when an aspect is added. In [5] a modular verification of an aspect relative to the specifications considered here is shown. Both of these techniques can be used as components in the checking of interference defined here, but the tool presented in [5] is used in this paper. Other work based on model checking determines whether the weaving of aspect scenarios is done correctly [8], or whether a full woven system is correct, using annotations that help construct the (non-modular) proof task for each weaving [10].

2. SPECIFICATIONS OF ASPECTS

A specification of an aspect consists of two parts: its *assumption* about any system into which it may reasonably be woven, and its *result assertion* (also called its *guarantee*) about properties of the result of weaving the aspect into any system satisfying the assumption.

The form of the specification is an instance of the *assume-guarantee* paradigm but generalized to relate to global properties of the system. The assumption of an aspect can include information on what is expected to be true at joinpoints, global invariants of the underlying system, or assumed properties of instances of classes or variables that may be bound to various parameters of the aspect when it is woven. (If the assumption states only properties required from the join-points of the aspect, then it can also be called the *precondition* of the aspect, otherwise the term *precondition* might be misleading.) The result assertion can include both new properties added by the aspect, and those properties of the basic system that are to be maintained in a system augmented with the aspect. Both parts of aspect specification are expressed in linear temporal logic.

Note that, as all the desired properties of the join-points are encapsulated by the assumption, the influence of the pointcuts on the behavior of the aspect is also hidden in it.

In general, there might be a large number of properties of the base system that should be maintained by the aspect. However, due to the analysis presented in [9], many of these properties do not need to be explicitly mentioned in the guarantee of the aspect. In that work ([9]) syntactically identifiable categories of aspects are presented. For each category of aspects broad classes of syntactically identifiable temporal properties preserved by all the aspects of this category are identified. Thus for each aspect and for each temporal property of the base system that should be maintained in the woven system as well, it is possible to efficiently decide whether or not this property should be explicitly included in the aspect's guarantee. Once we have stated, for example, that all safety properties of the base system are also true in the woven system, instances of such properties do not have to be listed in the guarantee. Treating such properties separately makes specification and verification of the rest of the properties much easier.

Definition 1. *An aspect is* correct *with respect to its assume-guarantee specification if, whenever it is woven (by itself) into a system that satisfies the assumption, the result will satisfy the guarantee.*

The question of determining "interference" between an aspect and a base system, i.e., whether or not the aspect maintains all the desired properties of the base system, and whether or not the aspect succeeds to ensure its guarantee, is taken care of in [5] and is out of scope of the current paper. In this paper we assume that all aspects are correct with respect to their specifications, and consider only possible interference among multiple aspects.

3. FORMALIZING SPECIFICATIONS, NON-INTERFERENCE AND PROOFS

We will first define semantic interference between two aspects and present a proof strategy for interference detection. Then it will be shown that performing these pairwise checks is enough to determine non-interference between any collection of aspects.

Given two aspects and their specifications, we can establish whether they interfere semantically, independently of any specific underlying system, while we rely on the correctness of the weaving process of the language in which the aspects are written. There might be some ambiguity in the weaving process, for example regarding the order of weaving aspect advices at a common joinpoint. However, here we assume a standard weaving strategy consistent with the aspect language used for implementation. First let us define system-dependent interference, and then define interference independent of any underlying system. (Recall that we assume that each aspect is correct with respect to its specification.)

DEFINITION 2. *Given two correct aspects A and B, and an underlying system S that satisfies the assumptions of both A and B, we say that* A does not interfere with B with respect to S *if the following property holds: Let S' be a system obtained from S by first weaving A into S, and then weaving B into the resulting system. Then in S' the guarantees of both A and B hold.*

DEFINITION 3. *Given two correct aspects A and B, we say that* A does not interfere with B *if for every system S satisfying the assumptions of both A and B, A does not interfere with B with respect to S.*

Notice the non-symmetry in the above definition of non-interference: if A does not interfere with B, it does not necessarily mean that B does not interfere with A, and vice versa.

Two aspects are semantically non-interfering if each does not interfere with the other in terms of Definition 3. Note that an aspect can change the values of variables used by the other even if they do not interfere, as long as the correctness of the specification is unchanged.

Let (P_A, R_A) be the specification of aspect A, where P_A is the Precondition of A (called the *assumption*), and R_A - its Result assertion (called *the guarantee*). In the same way, let (P_B, R_B) be the specification of B. Note that P relates to a system before the aspect has been woven into it, and R to that system augmented by the aspect, and each aspect is by itself correct. The correctness of the aspects is in terms of Definition 1, and it means that we assume that for every base system that satisfies P_A, the result of weaving A into this system satisfies R_A, and for every base system that satisfies P_B, the result of weaving B into this system satisfies R_B. Then for an underlying system S satisfying both assumptions (P_A and P_B), we need to prove that both sequential weavings - that of A before B and that of B before A - result in a system satisfying both guarantees (R_A and R_B).

DEFINITION 4. *In general, a set $\{A_1, \ldots, A_n\}$ of aspects is* interference-free *if whenever the assumptions $P_1, \ldots, P_n$ hold in a system, the augmented system obtained after weaving the aspects in any order satisfies the guarantees $R_1, \ldots, R_n$.*

Now let us describe non-interference and its proof more formally: The specifications of aspects we consider here are written in Linear Temporal Logic [15]. They relate to computations, which are sequences of states. (In fact, the definitions also hold for branching temporal logic, but the proof method does not.) The notation $S \models \psi$ is used to say that a temporal logic property ψ holds for every computation of a system S. Similarly, to say that a state predicate p holds at a state s of S we write $s \models p$.

To prove that A does not interfere with B, we need to show that

$$OK_{AB} = \forall S((S \models P_A \wedge P_B) \rightarrow ((S + A) + B \models R_A \wedge R_B))$$

holds, where by (S+A) we denote the result of weaving A into S (and by ((S+A)+B) - the result of weaving B into (S+A)). In the same way, to prove that B does not interfere with A we need to show

$$OK_{BA} = \forall S((S \models P_A \wedge P_B) \rightarrow ((S + B) + A \models R_A \wedge R_B))$$

Notice that these are two distinct statements, as in many cases the result of weaving A before B, $(S + A) + B$, will differ from the result of weaving B before A, $(S + B) + A$, as the advice of the previously woven aspect may not apply to the last-woven one. For the same reason both orderings above might differ from the result of simultaneous, AspectJ-like, weaving - the relation between them will be discussed in Section 7. The order of weaving will matter, for example, in the Composition Filters model [2], and in languages with dynamic aspects introduction. Moreover, even in AspectJ, if we first weave A into S and compile the program, and then weave B into the obtained Java bytecode, we do not get the same result as if A and B were woven into S at the same time by the AspectJ weaver.

There exists a way to prove the two above statements - OK_{AB} and OK_{BA} - directly, that will be described later, but it has several disadvantages. The following theorem will enable us to use an *incremental* step-by-step method:

THEOREM 1. *Let A and B be two aspects with the specifications (P_A, R_A) and (P_B, R_B) respectively, and assume that both aspects satisfy their specifications. Then to prove that A does not interfere with B it is enough to show that the following statements hold:*

$$KP_{AB} = \forall S((S \models P_A \wedge P_B) \rightarrow (S + A \models P_B))$$

*("*K*eeping the* P*recondition of B when weaving* A *before* B*") and*

$$KR_{AB} = \forall S((S \models R_A \wedge P_B) \rightarrow (S + B \models R_A))$$

*("*K*eeping the* R*esult of A when weaving* A *before* B*")*

Proof.
In other words, we need to prove that if A and B are correct aspects, and the KP_{AB} and KR_{AB} statements hold, then A does not interfere with B. The KP_{AB} statement means that the weaving of A into a system S satisfying the assumptions of both aspects does not invalidate the assumption of B. Such an S, in particular, satisfies the assumption of A. We know that after weaving A into a system S that satisfies P_A, R_A is true, for it is assumed as proven that A satisfies its specification. Thus together we have that (S+A) will satisfy not only the assumption of B, but also the guarantee of A, and the following statement will be true:

$$KP'_{AB} = \forall S((S \models P_A \wedge P_B) \rightarrow (S + A \models R_A \wedge P_B))$$

KR_{AB} means that weaving B into a system in which the guarantee of A holds does not invalidate this guarantee. B also satisfies its specification, so in the same way as for A, $S + B$ from KR_{AB} satisfies R_B, and we have

$$KR'_{AB} = \forall S((S \models R_A \wedge P_B) \rightarrow (S + B \models R_A \wedge R_B))$$

Now we can combine KP'_{AB} and KR'_{AB} by substituting S+A instead of S into KR'_{AB}. As a result we will obtain the desired property, OK_{AB}.
Q. E. D.

Symmetrically, to prove that the weaving of A can be performed after the weaving of B we need to show KP_{BA} and KR_{BA}, which, combined, will imply

$$OK_{BA} = \forall S((S \models P_A \wedge P_B) \rightarrow ((S+B)+A \models R_A \wedge R_B))$$

THEOREM 2. *Let $A_1, \ldots, A_n$ be aspects with the specifications $(P_1, R_1), \ldots, (P_n, R_n)$ respectively, and assume all these aspects satisfy their specifications. Assume also that for every pair of indices i, j $KP_{i,j}$ and $KR_{i,j}$ are true. Then the set $\{A_1, \ldots, A_n\}$ is interference-free.*

In order to prove the theorem, the following lemma will be useful:

LEMMA 1. *For every set of $n \geq 2$ aspects $\{A_1, \ldots, A_n\}$ satisfying their specifications $(P_1, R_1), \ldots, (P_n, R_n)$, if for every pair of indices i, j $KP_{i,j}$ is true, then for every base system S such that $S \models P_1 \wedge \ldots \wedge P_n$, the following holds: For every $0 \leq k < n$, $(\ldots(S + A_1) + \ldots + A_k) \models P_{k+1} \wedge \ldots \wedge P_n$ (where the case of $k = 0$ means that no aspects are woven into the system S).*

Proof (Lemma 1).
The proof is by induction on k.
Basis: $k = 0$. We need to show that $S \models P_1 \wedge \ldots \wedge P_n$, but this statement is one of the premises of the lemma.
Induction step: Assume that for every k such that $0 \leq k < m < n$ the statement holds, and let us prove it for $k = m$. Let S be a system such that $(S \models P_1 \wedge \ldots \wedge P_n)$. Let us denote the system $(\ldots(S + A_1) + \ldots + A_{m-1})$ by S'. We need to show that $(S' + A_m) \models P_{m+1} \wedge \ldots \wedge P_n$. From the premises of the lemma, for every $m + 1 \leq i \leq n$ the $KP_{m,i}$ property holds. Also, from the induction hypothesis, $S' \models P_m \wedge \ldots \wedge P_n$, and, in particular, $S' \models P_m \wedge P_i$. Together we have that indeed $(S' + A_m) \models P_i$ for every $m + 1 \leq i \leq n$.
Q. E. D.(Lemma 1)

Now let us prove Theorem 2. Let us be given a permutation $(i_1, \ldots, i_n)$ of indices. Without loss of generality, we can call them $(1, \ldots, n)$. (Clarification: We can always permute the aspects in the library so that for every j, aspect number i_j will stand on the j-th place. Then the order $1, \ldots, n$ on the permuted library will give the same sequence of aspects as the order $i_1, \ldots, i_n$ on the original one.) We need to prove that for every base system S, if $S \models P_1 \wedge \ldots \wedge P_n)$ then $(\ldots(S + A_1) + \ldots + A_n) \models R_1 \wedge \ldots \wedge R_n$. The proof is by induction on n.
Basis: If $n = 1$, there is only one aspect, A_1. Let S be a system satisfying P_1. The aspect A_1 satisfies its specification, thus the statement $(S \models P_1) \rightarrow (S + A_1 \models R_1)$ holds.
Induction step: We assume that the statement holds for any $1 \leq k < m$ aspects from the n given, and prove it for $k = m$. Let us be given a base system S satisfying $P_1 \wedge \ldots \wedge P_n$. We will denote by S' the system $(\ldots(S+A_1)+\ldots+A_{m-1})$. From the induction hypothesis we have that $S' \models R_1 \wedge \ldots \wedge R_{m-1}$. Lemma 1 is applicable here, so we also have that $S' \models P_m \wedge \ldots \wedge P_n$. In particular, $S' \models P_m$. Thus, as A_m is correct according to its specification, $S' + A_m \models R_m$. And for every $i \neq m$, $1 \leq i \leq n$, the $KR_{i,m}$ property holds, thus from the fact that $S' \models P_m \wedge R_i$ it follows that indeed $S' + A_m \models R_i$. Together we get that indeed $(\ldots(S + A_1) + \ldots + A_m) \models R_1 \wedge \ldots \wedge R_m$.
Q. E. D.

A proof that uses Theorem 1, that is, a proof that shows the assumption of the second woven aspect and the resulting assertion of the first one are preserved, is called an *incremental* proof. A *direct* proof merely shows that the weaving of the two aspects achieves both results.

Note that, as opposed to the incremental proofs assumed in Theorem 2, a direct proof of non-interference among pairs of aspects does not generalize to weaving of more than two aspects: even if aspects A, B, and C are pairwise interference-free, and are correct relative to their assumptions and guarantees, weaving of all three into a system with $P_A \wedge P_B \wedge P_C$ does not guarantee $R_A \wedge R_B \wedge R_C$ in the resulting system. For example, $S + A$ might not satisfy P_C, and thus, when B and C are woven, R_C might not result (even though just weaving C would give R_C). Thus the incremental method is essential for showing interference-freedom among groups of aspects of any size.

In some cases a conflict in the specifications of the aspects exists, which means that the specifications do not allow some composition of the aspects. Then for some order of weaving these aspects will always interfere, regardless of their advice implementation, as will be seen in Section 5.2. This composition of the aspects will be called *not feasible* according to the following definition:

DEFINITION 5. *Given two aspects A and B with specifications (P_A, R_A) and (P_B, R_B) respectively, the composition of A before B is* feasible *iff all the following formulas are satisfiable: $P_A \wedge P_B$, $R_A \wedge P_B$, $R_A \wedge R_B$*

If a composition of A before B is not feasible, it means that A has to interfere with B. Thus as a first step in detection of interference, a *feasibility check* can be performed - i.e., a satisfiability check on the appropriate formulas. It is recommended to perform a feasibility check before starting the full verification process described below, because this check is much easier and quicker, and then proceed to the verification only if the composition of the aspects is feasible. However, this is not an obligatory stage of the verification process, because if some contradiction exists, the verification method below will also detect interference and provide a counterexample.

4. PROOF IMPLEMENTATION USING MAVEN

In order to perform a verification without having to consider each possible underlying system, we use and adopt to our purposes the proof method suggested in [5] and the MAVEN tool presented there, while also improving MAVEN and making it more robust. The basic idea of that work, described for the verification of a single aspect relative to its specification, is that a single model can be generated from the aspect assumption, the pointcut description, and the advice, and used to model check the result assertion. If that model check succeeds, the augmented program resulting from the weaving of the aspect to any underlying system satisfying the aspect assumption is guaranteed to satisfy the result assertion of the aspect.

The single model to be checked is built from the tableau (state machine) that corresponds to the linear temporal logic assertion of the aspect assumption. This tableau is a generic model for all the systems satisfying the assumption of the aspect, and the state machine fragments that correspond to the advice are woven according to the pointcut descriptions. The tableau contains all the possible behaviors of the base systems into which the aspect can be woven. In other words, for any given underlying system that satisfies the assumption of the aspect, for every computation of this system there is a corresponding computation of the tableau, satisfying the same LTL properties. In [5] it is shown that the same holds for the woven systems: if the system S_1 results from weaving the aspect into some appropriate base system, and the system S_2 is the result of weaving the aspect into the tableau, then for every computation of S_1 there exists a corresponding computation of S_2. The properties we check are LTL properties, and an LTL property holds in a system iff each computation of this system, taken alone, satisfies this property. So if there exists a "bad" base system S such that S satisfies the assumption of the aspect, but the resulting assertion of the aspect is violated when it is woven into S, it means that there exists a "bad" computation in the woven system, violating the guarantee of the aspect, and for this bad computation there exists a corresponding bad computation in the (tableau + aspect) state machine. It follows that indeed it is enough to model-check the resulting assertion of the aspect on the (tableau+aspect) system only.

The state-machine fragments corresponding to the advice can be obtained from high-level code, e.g. in AspectJ or Java, using existing tools, such as Bandera [6]. Alternatively, the state machines can be created at earlier stages of the programming cycle to serve as abstract models of aspects during their design, before code is generated. Note that the aspect can consist of multiple pointcuts and advices, but not only the correctness of theorems 1 and 2 from Section 3 is not invalidated in this case, but this case is also supported by the implementation of the proof method presented below. However, if such complicated aspects do interfere, the diagnosis of the interference cause would be more efficient if the aspects were split into simpler ones, each with one advice only - if such a splitting is possible.

In order to use the MAVEN tool, we have to pose one restriction on the aspects: they should both be of the *weakly invasive* category, as defined in [9]. An aspect is weakly invasive if whenever its advice completes its execution, the resulting state, when the local variables and private objects of the aspect are ignored, already existed in the original underlying system. Notice that the advice can change state variables of the underlying system during its execution, and that local aspect variables can be modified with no restriction. The restriction to weakly invasive aspects is not a strong one, as most aspects fall into this category, including all of our examples. When an aspect is not weakly invasive, it may return control to the basic system in a state that was not previously reachable, and thus the effect of the base system's code is not limited by the assumption. The restriction on the aspects is not necessary theoretically: our statements are sound for all types of aspects. As MAVEN improves, or other verification tools for aspects become available, this restriction may become unnecessary.

Note also that a verification using MAVEN is relative to the standard weaving strategy built into that tool. Again, this is not inherent to our approach.

In order to use MAVEN as a subsystem for our technique, it was extended in several ways. Most significantly, - now it is possible to automatically determine whether the aspect verified is weakly invasive. In the first version of MAVEN the given aspect could be woven only into a tableau built from its assumption, while now it is possible to weave an aspect into an arbitrary transition system (in the NuSMV format). It also now allows initializing aspect variables globally, and preserving aspect values between activations of advice.

In our context, to show that A does not interfere with B, we will use the *incremental* method. The verification can be preceded by a feasibility check of the composition of aspects. The verification process, based on Theorem 1 and using the improved MAVEN tool, will be as follows:

1. To prove KP_{AB}, build a tableau that corresponds to the conjunction of the assumptions of the aspects, $P_A \wedge P_B$, weave the advice of A and show that the assumption of B, P_B, is true of the result.

2. In order to prove KR_{AB}, a tableau that corresponds to the conjunction of the assumption of B and the guarantee of A, $R_A \wedge P_B$, is built. Then after the advice of B is woven in, show that the guarantee of A, R_A, still holds for the result.

The proof that B does not interfere with A is symmetric. When we prove non-interference in both directions, although there are four steps of verification, we need to build only three different tableaus: one for the proofs of KP_{AB} and KP_{BA}, and the other two - for the proofs of KR_{AB} and KR_{BA}, respectively.

The above method is sound, due to Theorems 1 and 2, but not complete. There are two cases when the OK_{AB} check fails though the aspects do not interfere (the case of OK_{BA} is, of course, symmetric): The first case is a failure due to the incompleteness of the model-checking itself. If the model we are model-checking is infinite, or finite but too large, the model-checking will collapse without providing any answer. So, as always when model-checking is used, the models and the verified properties should be described at a sufficient level of abstraction. The second case is when the specification of some aspect is not the most general possible. Then there are two possibilities for failure - one arises when the assumption of aspect B, P_B, is not the weakest possible, and the other - when the guarantee of A, R_A, is not the strongest possible. In the first case, as P_B is not the weakest possible, it might happen that aspect A does not preserve the assumption of aspect B, but assures some other property, P'_B, that is enough for aspect B to operate correctly. Then the KP_{AB} check fails, but the OK_{AB} is true. In this case, if P_B was the weakest possible (i.e., such that $(S \models \neg P_B) \rightarrow (S + B \models \neg R_B)$), this possibility of interference would be eliminated. In the second case, symmetrically, it might happen that we can not prove that aspect B preserves the guarantee of A, because the assumption $R_A \wedge P_B$ is not strong enough to ensure R_A after B is woven, but the OK_{AB} property is true because A guarantees a stronger statement, R'_A, and with this assumption B is able to preserve R_A (for every system S, if $S \models R'_A \wedge P_B$, then $S + B \models R_A$). Notice some non-symmetry in the above statement - we have to assume R'_A, but can guarantee only R_A, because that is the property proven by the successful OK_{AB} check. In fact, by demanding that aspect B will preserve R_A when woven into *any* system that satisfies $R_A \wedge P_B$, we pose too strong a restriction, because we are interested in this statement only for base systems in which aspect A is present. This is an additional source of incompleteness in this case.

One could try to eliminate the second cause of incompleteness by verifying the OK_{AB} statement directly: Build a tableau that corresponds to the conjunction of the assumptions of the aspects, $P_A \wedge P_B$, weave the advice of A into the above tableau, and then weave the advice of B into the resulting state machine. Then verify both guarantees, $R_A \wedge R_B$, on the result. However, as we saw before, the direct method can not be generalized to more than two aspects. Moreover, even if we are interested only in detecting interference between two aspects, the analysis below shows that the space complexity of the direct method is higher than that of the incremental one, and thus the model-checker might fail to perform the direct verification, while succeeding to perform the incremental one.

For the complexity analysis, we assume, for convenience of presentation, that all the specifications are of (approximately) the same size, and all the advice machines are of (approximately) the same size as well. Given two aspects, A and B, with the specifications (P_A, R_A) and (P_B, R_B) respectively, we denote by r the maximal length of a formula from the aspect specifications $(max(|P_A|, |R_A|, |P_B|, |R_B|))$, and by M - the maximal size of the advice model of the aspects $(max(|M_A|, |M_B|))$.

Lemma 2. *The space complexity of the incremental method is* $O(2^{3r} \cdot M)$.

Proof.
The size of the tableau built from an LTL formula of length k is $O(2^k)$ (as shown in [3]). In our case the tableau is always built from two properties, so the size of the tableau is $O(2^{2r})$, and the size of the woven system on which the resulting assertion is model-checked is $O(2^{2r} \cdot M)$. When a formula of size k is verified on a model of size m, the space complexity of the model checking is $O(m \cdot 2^k)$ ([3]). In our case, $m = O(2^{2r} \cdot M)$ and $k = r$, so the altogether space complexity is $O(2^{2r} \cdot M \cdot 2^r) = O(2^{3r} \cdot M)$.
Q. E. D.

Lemma 3. *The space complexity of the direct method is* $O(2^{4r} \cdot M^2)$.

Proof.
Here first the assumptions tableau is built from the two assumptions of the aspects, and its size is $O(2^{2r})$. Then two advice models are woven into it, one after another, so the size of the woven system is $O(2^{2r} \cdot M^2)$ (M^2 appears here because there might be join-points of the second aspect inside the advice machine of the first). The property verified on this woven system is the conjunction of the two resulting assertions of the aspects, so the property size is $O(2^{2r})$. Together we have that the space complexity is $O(2^{2r} \cdot M^2 \cdot 2^{2r}) = O(2^{4r} \cdot M^2)$.
Q. E. D.

5. EXAMPLES

5.1 Encrypting Passwords

In this example we discuss two reusable aspects, E and F, that may appear in a security-aspects library and might be used in a password-protected system. An example of such a system can be the internet terminals of a bank, providing the possibility of viewing and/or updating the user's account via the internet.

Aspect E is responsible for encrypting the passwords before sending.The joinpoint E advises is the moment when the password-containing message is to be sent from the login screen. E's advice is a "before" advice, that encrypts the message. E should guarantee that each time a password is sent, it is encrypted, and the assumption of E might be that password-containing messages are sent only from the login screen in the base system. In fact, there is more to E: each time a password is received, it is be decrypted. But this part is irrelevant to our example, so we'll ignore it here. A possible specification for E can thus be that E assumes that password-containing messages are sent only from the login screen in the base system, and guarantees that each time a password is sent, it is encrypted. More formally it can be written as follows:

$$P_E = G(\mathit{psw\_send} \leftrightarrow \mathit{login\_psw\_send})$$

and

$$R_E = G(\mathit{psw\_send} \rightarrow \mathit{encrypted\_psw})$$

where the predicate *psw_send* means that a message containing a password is being sent, and *login_psw_send* means that the password is being sent from the login screen. The assumption of the aspect might seem arbitrary, but this choice was guided by a possible implementation of the aspect. It might be the case that the aspect is unable to identify password-containing messages from the message content only, and then the pointcut could be defined as the creation of a message containing information from some specific field of the graphic user interface. Note that the aspect is generic (to enable reuse), and thus the field from which the information is taken should be a parameter bound to the aspect when adding the aspect to a concrete system.

Aspect F is responsible for treating a situation when the user forgets the password. Usually in password-guarded systems there is a way of retrieving your password once you forget it. F provides a list of security questions to the user, and if the questions are answered correctly, F guarantees that the user will get his password via an e-mail. In order to add this functionality to the system, F should add some introductory operation. For example, a new button - "Forgot my password" - can be added to the system so that we can define the pointcut of F as the moment when this button is pressed. F's advice then provides the dialog with questions, checks the answers and in case all the answers are correct - sends an e-mail to the user. The button is added by F itself, thus one of the possible ways to specify F is to say that F assumes nothing about the system (because F itself adds to the system the possibility to report forgetting the password), and guarantees that whenever the security check is passed, the forgotten password will be sent to the user (and if the check is never passed, the password remains forgotten forever as it was in the base system). More formally, F's assumption is

$$P_F = true$$

And F's guarantee is

$$R_F = [G((button\_pressed \wedge quest\_answered) \rightarrow F(psw\_send)))]$$

where *button_pressed* is the flag that means forgetting the password has been reported and not yet treated.

Let us check the possibility of sequential weavings. F's assumption is *true*, thus E can not violate it. Thus in order to check the possibility of weaving F after E, we need to prove only that the weaving of F maintains the guarantee of E (the KR_{EF} statement):
$\forall S((S \models G(psw\_send \rightarrow encrypted\_psw)) \rightarrow$
$(S + F \models G(psw\_send \rightarrow encrypted\_psw\ )))$
This statement seems to be reasonable, and the feasibility check succeeds, but the advice of aspect F is implemented in such a way that the password sent from it is not encrypted. Thus when trying to verify the KR_{EF} statement, a counterexample is obtained. It is a computation in which at some state s_1 the predicate *button_pressed* became true, and at the same time the predicate *encrypted_psw* was false. Two states after that, at a state s_2, due to the operation of the aspect F, *quest_answered* became true (while *button_pressed* was still true), and in the next state, s_3, *psw_send* became true. But F does not encrypt the passwords, thus *encrypted_psw* was still false at s_3, contradicting the implication in R_E, so the verification failed.

In order to check the possibility of weaving E after F, we need to prove that the weaving of F to a system satisfying both assumptions maintains the assumption of E (the KP_{FE} statement):
$\forall S((S \models G(psw\_send \leftrightarrow login\_psw\_send) \wedge (true)) \rightarrow$
$(S + F \models G(psw\_send \leftrightarrow login\_psw\_send)))$.
However, the implementation of the advice of F leads to violation of the assumption of E, because F does not send the password from the login screen. Note that in this case, again, there is no contradiction in the specifications of E and F, so the feasibility check succeeds, and the interference is detected during the verification only.

Two remarks about the example: (1) If E and F were the only two aspects existing in the library, it would be very easy to detect the interference just by looking at the library, but, as already noted, in real life many different aspects are added to systems and libraries by different groups of people, and an automated solution is needed. (2) In this example we see that the conflicting aspects do not share any joinpoints, and the interference doesn't emerge from updating common variables.

A variant of this example, both as Java code and abstract models, is presented at `http://www.cs.technion.ac.il/ssdl/pub/SemanticInterference/`
It includes the input for verification by the indirect method, checking whether E interferes with F, including aspect descriptions in the NuSMV format, and appropriate LTL assertions (the KP_{EF} stage is not interesting in our case, so only the KR_{EF} stage is presented), and the output of the verification - the counterexample provided by NuSMV. Some statistics for this variant, comparing the verification by indirect and by the direct method, appear in Figure 1. $|M.|$ means the model size and is measured by the number of BDD nodes in the model, and $|Ex.|$ means the number of states in the counterexample found (0 means the result of verification was *true*). These statistics show the additional disadvantage of the direct method: not only is this method applicable only for the case of two aspects, but also the models it creates for verification of two aspects interference are much bigger than those created by the incremental method. In average for this example the models created by the direct method are more than 4 times bigger than those created by the incremental, and the maximal ratio of model sizes for this example is almost 7.

Remark: as a result of verification of KP_{FE}, a counterexample was obtained. Thus it would be possible to stop the verification at this stage and try to amend the aspects and/or their specifications before continuing to verification of KR_{FE}.

Direct method			**Incremental method**		
Check	$\|M\|$	$\|Ex\|$	**Check**	$\|M\|$	$\|Ex\|$
OK_{EF}	7778	18	KP_{EF}	1127	0
			KR_{EF}	1283	12
OK_{FE}	8700	18	KP_{FE}	2375	12
			KR_{FE}	2450	0

Figure 1: Security example statistics

5.2 ATM Communication and Card Theft

In this example we consider two aspects that can be used in a system with remote authorized access. They are most useful for systems in which each user can have only one open session at a time. The first aspect (aspect C below) treats communication failures in the system, that occurred during authorization process or while some authorized user was logged in. Its goal is to assure that the user will be able to log in again after the communication is restored. The second aspect (aspect T) prevents identity-theft: for example, if a wrong password is provided in several consequent attempts of logging in, the aspect guarantees that the user is blocked. One possible system in which these aspects might be used is an ATM system of a bank, consisting of several ATM machines and a server. The user interacts with this system by first inserting a card and code, and then, if permission is granted, entering a request for some bank operation (money withdrawal, or account balance check).The ATM machine communicates with the server to process user requests, and the server grants or denies permission to perform operations, and processes the operations permitted. From the point of view of the aspects, the card serves as a user-login, and code - as a password. To make the example more intuitive, all the descriptions below are written in terms of the ATM system (note that the aspects are still general and reusable, even after this concretization, because many different implementations of the above-described ATM system are possible). A more detailed description of the aspects is as follows:

Aspect C (for *C*ommunication) is responsible for treating communication failure between the server and an ATM machine. In case of communication failure, the aspect checks whether there is a card stuck in the ATM machine, and returns it to the user. One of the reasonable specifications of C is: C assumes that the only case when a card can get stuck in a machine is when a communication failure occurred while the card was in the machine. In such a case C guarantees that a card is never stuck in a machine forever. Formally,

$$P_C = G(card\_in \rightarrow F(\neg card\_in \vee comm\_fail))$$

(which means that if a card was inserted, either it will be eventually returned, or a communication failure - indicated by $comm\_fail$ predicate - will happen), and

$$R_C = G(card\_in \rightarrow F(\neg card\_in))$$

(which means that if a card was inserted, it will eventually be returned). Note that the $comm\_fail$ predicate does not necessarily represent a general communication failure in the whole system. Our abstraction here is that the flags $comm\_fail$ and $card\_in$ relate to a communication failure and card status at a particular ATM.

Another aspect, T (for *T*heft), comes to prevent card-theft. A possible specification is: T assumes that there exists a possibility to detect that the card is stolen, and if the card is stolen it will remain stolen forever. T ensures that such a card will never return to the user. Formally,

$$P_T = G(card\_stolen \rightarrow G(card\_stolen))$$

and

$$R_T = G((card\_in \wedge card\_stolen) \rightarrow G(card\_in))$$

Let us examine the possibility of sequential weaving of T after C. One of the statements we need to show is that T does not violate the guarantee of C,KR_{CT}:
$\forall S((S \models (G(card\_in \rightarrow F(\neg card\_in)) \wedge (card\_stolen \rightarrow G(card\_stolen)))) \rightarrow (S{+}T \models G(card\_in \rightarrow F(\neg card\_in))))$.
There is a contradiction in the requirement from $S+T$: On one hand, we require R_C, that says that an inserted card will eventually be returned in every case. On the other hand, T satisfies its specification, and P_T was true in S, thus R_T should be true in $S+T$. By that we require that in some special case (that of a stolen card) the card will never be returned, which contradicts our first requirement (which is the R_C assertion). This contradiction is found when the R_C assertion is model-checked on the Tab+T system, built by weaving T into the tableau Tab of $R_C \wedge P_T$. Thus it is impossible to weave T after C, at least not with such a specification. Note that in this case the feasibility check is also able to detect interference, because there is a contradiction between R_C and R_T. Similarly, we will find a contradiction and counterexample to the weaving of C after T.

Verification statistics for this variant, comparing the verification by indirect and by the direct method, appear in Figure 2. Note that here, as in the previous example, the model sizes of the incremental method are much smaller than those of the direct (about $\frac{1}{3}$ of their size).

Direct method			**Incremental method**		
Check	$\lvert M\rvert$	$\lvert Ex\rvert$	**Check**	$\lvert M\rvert$	$\lvert Ex\rvert$
OK_{CT}	3154	8	KP_{CT}	1038	0
			KR_{CT}	776	8
OK_{TC}	3045	5	KP_{TC}	1028	8
			KR_{TC}	1098	11

Figure 2: ATM example statistics

6. ERROR ANALYSIS

When interference has been detected between two aspects, the cause of the verification failure should be localized - which property was violated, and which advice is "guilty". The verification process is divided into stages, making the localization straightforward: if we fail to prove the OK_{AB} and there is a problem in violating the assumption of B, the proof of KP_{AB} will fail, and if the advice of B violates the guarantee of A, the failure will occur in the proof of KR_{AB}.

After the cause of the failure is localized, one needs to decide on what steps should be taken next. In many cases there is a need to add the functionality of both aspects to the base system, in spite of the interference detected between them. There are several possible ways to handle this problem, depending on the type of the interference detected, and the results of the feasibility check (thus it is recommended to perform the feasibility check of the specifications as a first step of error analysis in case an interference is detected). One should then decide whether to change the advice of one of the aspects (or both), and whether the specification of the aspects should be refined. For the examples from Section 5, the way to interference elimination might be as follows:

For the example in Section 5.2, the weaving of C before T appears to be non-feasible, thus the first step to elimination of the interference is to try and refine the specification of the aspects in such a way that the composition becomes feasible. And indeed, there is a possibility of such a refinement: if we knew of the possibility of stealing the card, or, more generally, of special events other than communication failure that can cause the card to be stuck, we could update the guarantee of C to treat these events: $R_C = G((card\_in) \rightarrow F(special\_event \vee \neg card\_in))$, and then add $(card\_stolen \rightarrow special\_event)$ to P_T. Then C would not interfere with T. Note that if such a refinement is possible, it means that the specification provided by the user for one of the aspects (or, maybe, for both) was too strong.

For the example in Section 5.1, on the other hand, aspect F interferes with E, though the composition of E after F is feasible. In this example it is impossible to eliminate the interference by changing only the specification of the aspects, and a change in one advice, or in both, is necessary. For instance, we can change the advice of F to bring the user to a version of the login screen where the password can be changed, instead of sending the e-mail with the password. In this case, if E is woven after F, the password-sending operation of F is done by the user as another login-password send and thus will be a legal joinpoint of E. Therefore the advice of E will be performed and no password will be sent unencrypted. More formally: the specification of F can stay the same, but as a result of the change in the advice, whenever *psw_send* is true, so is *login_psw_send*. Aspect E and its specification will stay as before. Now the verification will be of F's new code relative to the specifications, so that KP_{FE} and KR_{FE} now will hold. This means that the sequential weaving of first F and then E is possible. Notice, however, that weaving first E and then F would still be problematic.

7. JOINT WEAVING

The above discussion treated only sequential weaving. Let us now consider the case of simultaneous weaving. Such a weaving at every point of the program decides whether to apply A, or B, or both, and in which order (as opposed to sequential weaving, where the possibility of inserting only one aspect at a time is checked). One approach is to reduce joint weaving to sequential weaving, whenever possible. Then given aspects A and B, we would like to check whether weaving both A and B together into some base system is equivalent to one of the sequential weavings (A after B or B after A) into the same base system. If A and B have a common join-point, then the ordering of application may not be well defined, and this is well-known to create possible ambiguity. The lemmas below assume no common join-points, because some of the alternative semantic meanings violate the lemmas.

The following definitions will be useful to us:

Definition 6. *Let S be a system, and A and B - two aspects. Let us denote by J the set of all the joinpoints that are matched by B in S, and by J' - the set of all the joinpoints that are matched by B in (S+A). We say that A* creates a joinpoint *matched by B if there exists a joinpoint $j_1 \in J'$ such that j_1 is not in J (that is, J' is not included in J). We also say that A* removes a joinpoint *of B if there exists a joinpoint $j_2 \in J$ such that j_2 is not in J' (that is, J is not included in J')).*

Thus if A does not create or remove joinpoints matched by B, it means that the joinpoints matched by B in the original system S are exactly the same as in (S+A) - the system obtained by weaving A into S.

The following lemma shows that if weaving aspect B into a base system does not affect join-points of A (i.e, the joinpoints of A in the woven system are the same as in the base one), and the symmetric statement holds - weaving aspect A into a base system does not affect join-points of B - then the order of weaving of the aspects "does not matter" for the final result:

Lemma 4. *Let S be a system such that there is no joinpoint in S matched by both A and B, and they do not create or remove joinpoints matched by each other. Then the simultaneous weaving of A and B into S (S+(A,B)) is equivalent to both sequential weavings: of A before B ((S+A)+B) and of B before A ((S+B)+A). That is, the weaving is both associative and commutative.*

It is also not difficult to treat the possibility of adding joinpoints of the second woven aspect in the advice code of the first, as seen in the following lemma.

Lemma 5. *Let S be a system such that there is no joinpoint in S matched by both A and B, and B does not create or remove joinpoints matched by A. Let it be possible for A to create joinpoints matched by B, but only inside its (A's) own advice and without removing joinpoints matched by B. Then the simultaneous weaving of A and B into S (S+(A,B)) is equivalent to weaving A before B ((S+A)+B). That is, the weaving is associative, but not necessarily commutative.*

The proofs of the lemmas appear at the same site as the example. In order to check that the above lemmas can be applied, we need to establish that A and B do not match common joinpoints. For that purpose existing tools mentioned in Section 1 can be used ([4, 7]).

8. CONCLUSIONS

In this paper we have defined semantic interference among aspects relative to their specifications and shown an effective way to detect interference or prove interference-freedom of multiple aspects in a library.

The interference-detection method is modular: the library of aspects is checked independently of any base system. Thus when the user would like to weave multiple aspects from the library into some base system, the only check that should be performed is that the base system satisfies the assumptions of all the aspects that will be added to it.

The result of the verification process is not a "yes" or "no" answer, stating whether or not the current library is interference-free: the results of the verification are more informative. For each aspect we know with which aspects it does not interfere, and also for every aspect with which an interference exists, we know what is the cause of the interference, and in which order of weaving it occurs. All this information can serve as usage guidelines for the developers who would like to use aspects from the verified library. In case the library as a whole is not interference-free, a developer might chose some interference-free subset of the library (recall that pairwise interference-freedom of the aspects in any set is enough to guarantee interference-freedom of the set as a whole), or decide on an appropriate weaving order of the aspects to prevent interference.

There already exist libraries of reusable aspects. One of them - a library implementing ACID properties for transactional objects - is described in [13], and different kinds of interference among the aspects from this library are mentioned there. As future work we would like to apply our verification method to detect interference among aspects from that library, and analyze the different types of interference described in that paper.

Currently we have started to work on an aspect case study demonstrator of the AOSD-EUROPE project, based on the Toll System [19] written by the Siemens company: a system designed for computing fees and charging the drivers for the use of toll roads. The goals of the case study are formalization and verification of aspects from the Toll System and include detection of possible interference among them. As one example, we have discovered and are analyzing interference between an aspect to impose fines and an aspect to give discounts on the use of the toll road.

This paper deals with interferences between the aspects, but the formalization and proof methods it provides can be easily extended to treat some other types of aspect interactions that can be formalized and checked by similar means. For example, aspects may work cooperatively when one aspect is dependent on another to establish its assumption and cannot be woven into a system unless that other aspect is also there.

9. REFERENCES

[1] D. Balzarotti, A. D'Ursi, L. Cavallaro, and M. Monga. Slicing AspectJ woven code. In *Proc. of Foundations of Aspect Languages Workshop (FOAL05)*, 2005.

[2] L. Bergmans and M. Aksit. Composing crosscutting concerns using composition filters. *CACM*, 44:51–57, 2001.

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

[4] R. Douence, P. Fradet, and M. Sudholt. Composition, reuse, and interaction analysis of stateful aspects. In *Proc. of 3th Intl. Conf. on Aspect-Oriented Software Development (AOSD'04)*, pages 141–150. ACM Press, 2004.

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

[6] 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.

[7] W. Havinga, I. Nagy, L. Bergmans, and M. Aksit. A graph-based approach to modeling and detecting composition conflicts related to introductions. In *AOSD '07*, pages 85–95. ACM Press, 2007.

[8] E. Katz and S. Katz. Verifying scenario-based aspect specifications. In *Proc. Formal Methods: International Symposium of Formal Methods Europe (FM05)*, volume 3582 of *LNCS*, pages 432–447. Springer, 2005.

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

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

[11] R. Khatchadourian and N. Soundarajan. Rely-guarantee approach to reasoning about aspect-oriented programs. In *Proc. of Software Engineering Properties of Languages and Aspect Technologies SPLAT'07*, 2007.

[12] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *Proc. ECOOP 2001*, LNCS 2072, pages 327–353, Jun 2001. `http://aspectj.org`.

[13] J. Kienzle and S. Gélineau. AO challenge - implementing the ACID properties for transactional objects. In *Proc. 5th International Conference on Aspect-Oriented Software Development (AOSD 2006)*, pages 202–213. ACM, 2006.

[14] 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.

[15] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1991.

[16] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319–340, 1976.

[17] 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.

[18] M. Storzer and J. Krinke. Interference analysis for AspectJ. In *Proc. of Foundations of Aspect Languages Workshop (FOAL03)*, 2003.

[19] Toll system demonstrator. `http://www.aosd-europe.net` (under the "Industry" section).

[20] J. Zhao. Slicing aspect-oriented software. In *IEEE International Workshop on Programming Comprehension*, pages 251–260, 2002.