# Aspects and Modular Reasoning in Nonmonotonic Logic

Klaus Ostermann
Darmstadt University of Technology, Germany
ostermann@informatik.tu-darmstadt.de

## ABSTRACT

Nonmonotonic logic is a branch of logic that has been developed to model situations with incomplete information. We argue that there is a connection between AOP and nonmonotonic logic which deserves further study. As a concrete technical contribution and "appetizer", we outline an AO semantics defined in default logic (a form of nonmonotonic logic), propose a definition of modular reasoning, and show that the default logic version of the language semantics admits modular reasoning whereas a conventional language semantics based on weaving does not.

## 1. INTRODUCTION

There has been a lot of debate in the aspect-oriented community on how aspects influence program understanding or reasoning about programs, in particular how aspects influence "modular reasoning" (e.g., [12, 8]) (although modular reasoning has never really been defined). Previous works have concentrated on restricting AO languages in order to ease modular reasoning (e.g., [9, 2]). In this paper, we investigate a different approach: Rather than restricting the language, we propose to use a different reasoning model, namely nonmonotonic reasoning (see [3] for an overview).

In classical (monotonic) logic, adding a piece of information to a knowledge base never reduces the set of its consequences. Intuitively, monotonicity indicates that learning a new piece of knowledge cannot reduce what was previously known. Nonmonotonic logics (the formal incarnations of nonmonotonic reasoning) have been developed to deal with incomplete and changing information. Nonmonotonic logic allows to revise conclusions if new knowledge arrives, and provides rigorous mechanisms for taking back conclusions that no longer fit to newly learned knowledge, and deriving new, alternative conclusions instead.

In this paper, we argue that there is a fruitful connection between nonmonotonic logic and aspects. Using nonmonotonic logic, it is possible to specify the semantics of an AO language with pointcuts and advice in a very direct and compositional way: no kind of weaving or other global operation needs to be build into the semantic definitions. The absense of any operations requiring global knowledge means that reasoning with local knowledge is also easier. To validate this claim, we propose a definition of modular reasoning and show that nonmonotonic logic restores the ability for modular reasoning, albeit at the cost of giving up monotonicity.

The rest of the paper is structured as follows. In the next section, we give a very short introduction to default logic. In Sec. 3, we give a semantics of an AO language with pointcut and advice based on default logic and compare it with a conventional AO language semantics based on weaving. In Sec. 4 we consider the problem of modular reasoning and discuss how nonmonotonic logic influences modular reasoning. Sec. 5 discusses variants of default logic that employ priorities, and how these variants can be used to model advice precedence rules. Sec. 6 discusses what has been achieved.

## 2. DEFAULT LOGIC

A typical example in nonmonotonic logic is that we know birds usually fly, and that Tweety is a bird, and hence conclude that Tweety flies - until we learn that Tweety is actually a penguin. Using default logic [22] - one particular variant of nonmonotonic logic - we can formalize this situation as follows:

$$\frac{\mathit{bird}(X)\ :\ \mathit{flies}(X)}{\mathit{flies}(X)}$$

This rule is a so-called *default*, and can be read as "If X is a bird, and if it is consistent to assume that X flies (that is, it cannot be concluded that X does *not* fly), then conclude that X flies". In general, a default $\delta$ has the form $\frac{\varphi:\ \psi_1,...,\psi_n}{\chi}$, where $\varphi, \psi_1, ..., \psi_n, \chi$ are predicate logic formulae, and $n > 0$. The formula $\varphi$ is called *prerequisite*, the part to the right of the colon, $\psi_1, ..., \psi_n$ *justifications*, and the part below the bar, $\chi$, is the *consequent*. A default is *applicable* to a deductively closed set of formulae $E$, if $\varphi \in E$ and $\neg\psi_1 \notin E, ..., \neg\psi_n \notin E$.

In general, the set of conclusions that we can draw from a knowledge base with defaults is not unique. For example, if we know that members of the green party typically do not like cars, and members of an automobile club usually like cars, and John is members of both green party and automobile club, then we can conclude both that John likes cars and that he does not like cars.

This seeming chaos is ordered by so-called *extensions* - possible world views based on the given defaults. Technically, an extension is a superset of the knowledge base that

*Sixth International Workshop on Foundations of Aspect-Oriented Languages (FOAL 2007),* March 13, 2007, Vancouver, BC, Canada.
Copyright 2007 ACM ISBN 1-59593-671-4/07/03 ...$5.00.

$$\frac{\vec{a} = \mathit{ApplicableAdvice}(o,m)}{...o.m(\vec{v}) \hookrightarrow ...o.m[\vec{a}](\vec{v})} \qquad (\textsc{Weave})$$

$$\frac{\mathit{AdviceLookup}(a) = (\vec{x}, e)}{...o.m[a,\vec{a}](\vec{v}) \hookrightarrow ...e\left[{}^{o}/_{\mathbf{this}}, {}^{\vec{v}}/_{\vec{x}}, {}^{o.m[\vec{a}](\vec{v})}/_{\mathsf{proceed}}\right]} \qquad (\textsc{AdvExec})$$

$$\frac{\mathit{MethodLookup}(o,m) = (\vec{x}, e)}{...o.m[\emptyset](\vec{v}) \hookrightarrow ...e\left[{}^{o}/_{\mathbf{this}}, {}^{\vec{v}}/_{\vec{x}}\right]} \qquad (\textsc{MethExec})$$

**Figure 1: AO language semantics in the style of Jagadeesan et al**

is consistent and closed under deduction and application of defaults [22]. In the example, we would have two distinct extensions, in which John likes and does not like, respectively, cars. A large part of the theory of default logic is concerned with the existence and construction of extensions and the relations between different extensions.

Reiter's original definition of extensions is a non-constructive fixed point equation based on the above properties, but we give an equivalent operational definition based on [18, 3, 4]. For this purpose, we define a *default theory* to be a pair $T = (W, D)$ consisting of a set $W$ of predicate logic formulae (sometimes called *background theory*) and a countable set of defaults $D$. The extensions of $T$ are the deductive closures of all sets $E$ that can be generated by the following non-deterministic algorithm[1]:

```
$E := W; A := \emptyset;$
while there is a default $\delta \notin A$ that is applicable to E {
  $E := E \cup \{consequent(\delta)\}; A := A \cup \{\delta\};$
}
if $\forall\delta \in A.E$ is consistent with all justifications of $\delta$
  then return $E$ else failure
```

The algorithm first uses applicable defaults in an arbitrary order to build a candidate for an extension. The consistency check in the last two lines then checks whether $E$ is really an extension. In general, extensions are neither unique (due to the non-deterministic choice of the next default) nor need to exist at all (due to the consistency check).

It may look strange that every default is applied at most once in the algorithm. This is sufficient, because the rule about birds above is technically not a default but a *default schema* since it contains a free variable (namely $X$). Default schemata are implicitly interpreted to mean the set of defaults $\frac{\varphi\sigma:\ \psi_1\sigma,\ldots,\psi_n\sigma}{\chi\sigma}$ for all ground substitutions $\sigma$ that assign values to all free variables in the schema. For example, if we have two birds Tweety and Trixy, then our default schema creates two separate defaults $\frac{bird(Tweety)\ :\ flies(Tweety)}{flies(Tweety)}$ and $\frac{bird(Trixy)\ :\ flies(Trixy)}{flies(Trixy)}$.

## 3. ASPECTS AND DEFAULT LOGIC

Usually, pointcuts are implemented by static or dynamic weaving (that is, code transformation) or by interception and dynamic lookup. This view is also reflected in most formal accounts of AOP languages. Let us consider an object-oriented language with "around" advice that can advise method calls. In Jagadesaan et al's calculus of aspect-oriented programs [11], the (small-step) operational semantics rules for method lookup look roughly as sketched in Fig. 1. We leave out many details that are irrelevant for the purpose of this paper. The "..." part in the transition rules stands for dynamic entities of the operational semantics, such as call stacks or heaps. We also refrain from showing all the other rules of a complete operational semantics, since the rules for method and advice execution are sufficient to illustrate our idea.

[1]Recall the definition of *applicable* on the previous page

A method call $o.m(\vec{v})$ is executed by first looking up all advice that applies to a method call and sorting the advice in some order (inside the *ApplicableAdvice* function, whose definition is not shown here), and weaving the sorted list of advice $\vec{a}$ into the method call (WEAVE). This weaved method call is then executed by taking the first advice from the list, looking up the formal arguments and body of the first advice, substituting **this** and the formal parameters $\vec{x}$ by the receiver object and the actual parameter values, respectively, and substituting `proceed` by a method call that removes the first advice from the list of pending advice (ADVEXEC). If no advice is left, the original method body is executed (METHEXEC). In both cases, the lookup functions return a list with the names of the formal parameters and the advice/method body.

Let us now study how we could encode a similar AO language semantics using default logic. We propose rules as presented in Fig. 2. The meaning of a method call is now a bit different: Whereas in Fig. 1 an expression $o.m[\vec{a}](\vec{v})$ denotes a method call where the execution of all advice in $\vec{a}$ is *pending*, we now interpret it to mean a method call where all advice in $\vec{a}$ *have already been executed.* Hence we do not need a separate syntactic form $o.m(\vec{v})$ for method calls before weaving; rather, normal method calls are denoted as $o.m[\emptyset](\vec{v})$.

There are only two computation rules, (METH) and (ADV). Due to the different meaning of the advice list in method calls, (ADV) *adds* rather than removes the name of the executed advice to the method call that replaces `proceed`. There is no weaving rule anymore. Rather, the behavior of (METH) and (ADV) is controlled by the auxiliary predicates *NextAdvice* and *unadvised*, which are defined using defaults. If there is no information to the contrary, we assume that a method call is unadvised (UNADV). If however, there is some applicable advice $a$ that has not yet been executed, and if it is consistent to assume that it is the next advice to execute, then we conclude that $a$ will be the next advice (NEXTADV). Furthermore, a call with applicable advice is not unadvised (SOMEADV).

To avoid that two different advice are both simultaneously the next one, we implicitly assume the existence of the usual inference rules of equality, in particular $\frac{x \neq x'}{\neg(x = x')}$.

$$\frac{\begin{array}{c}\mathit{MethodLookup}(o,m) = (\vec{x}, e)\\ \mathit{unadvised}(o,m,\vec{a})\end{array}}{...o.m[\vec{a}](\vec{v}) \hookrightarrow ...e\left[{}^{o}/\mathbf{this}, {}^{\vec{v}}/_{\vec{x}}\right]} \quad (\textsc{Meth})$$

$$\frac{\begin{array}{c}\mathit{NextAdvice}(o,m,\vec{a}) = a\\ \mathit{AdviceLookup}(a) = (\vec{x}, e)\end{array}}{...o.m[\vec{a}](\vec{v}) \hookrightarrow ...e\left[{}^{o}/\mathbf{this}, {}^{\vec{v}}/_{\vec{x}}, {}^{o.m[a,\vec{a}](\vec{v})}/\mathsf{proceed}\right]} \quad (\textsc{Adv})$$

$$\frac{\mathit{true} \ : \ \mathit{unadvised}(o,m,\vec{a})}{\mathit{unadvised}(o,m,\vec{a})} \quad (\textsc{Unadv})$$

$$\frac{a \in \mathit{ApplicableAdvice}(o,m) \wedge a \notin \vec{a} \ : \ \mathit{NextAdvice}(o,m,\vec{a}) = a}{\mathit{NextAdvice}(o,m,\vec{a}) = a} \quad (\textsc{NextAdv})$$

$$\frac{a \in \mathit{ApplicableAdvice}(o,m) \wedge a \notin \vec{a}}{\neg \mathit{unadvised}(o,m,\vec{a})} \quad (\textsc{SomeAdv})$$

**Figure 2: AO language semantics using default logic**

To appreciate the difference between default and classical logic, assume for a moment that the colons in Fig. 2 would be replaced by conjunction operators (i.e., we would use classical rules). In this case, we could never prove a goal of the form $\mathit{unadvised}(o,m,\vec{a})$ or $\mathit{NextAdvice}(o,m,\vec{a}) = a$ because the same goal that we want to prove also appears in the premise of its rule. Hence the semantics would be useless. Similarly, if we would just remove the justifications, the semantics would be useless because we could prove $\mathit{unadvised}(o,m,\vec{a})$ for arbitrary $o$, $m$, and $\vec{a}$.

Now, the question arises whether the default theory in Fig. 2 has any extensions, and if any, what they look like. Luckily, all default rules in Fig. 2 are so-called *normal defaults*, meaning that the justification is the same as the consequent. Normal default theories are particularly well-behaved. Besides other important properties, normal default theories *always* possess extensions [22], which answers the first question.

Is there only a unique extension? No - in case more than one advice is applicable at some point, there is more than one extension, namely one for every possible advice execution order. This reflects the fact that there is no a-priori order among different overlapping advice. The difference to previous approaches is that we can now deal with this situation *within our reasoning framework*, and study the ambiguity in terms of extensions.

Let us now analyze informally to which degree the two language semantics agree with each other. If at most one pointcut applies at any joinpoint, the two semantics agree because in this case, there is only one unique extension in the default theory, which is the same theory that is generated by the conventional operational semantics. The semantics differ in how they treat shared joinpoints (more than one pointcut applies). In Fig. 1, the *ApplicableAdvice* lookup function orders all applicable advice in a specific order, whereas in Fig. 2 every potential execution order is represented by a different extension. We will later discuss how variants of default logic such as *prioritized default logic* [6] can be used to model global orders or ordering hints (such as *declare precedence* in AspectJ) on advice.

## 4. MODULAR REASONING

We will now attempt to give a semi-formal definition of modular reasoning. Reasoning can be performed with respect to a knowledge base, whereby we define a knowledge base as a set of logic formulae (or axioms) $F$ in the case of classical reasoning, and as a default theory $T = (W, D)$ in the case of default reasoning. What can be concluded from the knowledge base is the deductive closure of $F$ in the classical case, and the set of extensions of $T$ in the default logic case.

We view the “partial evaluation” of the operational semantics rules with the current program as the knowledge base which we use to reason about the operational behavior of a program. By this we mean the set of rule instances where all meta-variables that refer to parts of the program are replaced by ground substitutions from the program. Recall that the operational semantics inference rules are actually rule schemata that stand for a set of rule instances, hence we can talk about the set of rule instances $\mathit{ruleinstances}(P)$ for a given program $P$[2]. For example, if our program contains an object *anObj* and a method *aMeth* of this object whose body returns **this**, then

$$\frac{\mathit{Unadvised}(\mathit{anObj}, \mathit{aMeth}, \emptyset)}{...\mathit{anObj}.\mathit{aMeth}[\emptyset]() \hookrightarrow ...\mathit{anObj}}$$

is a rule instance of (Meth).

Please note in this context that the lookup functions *AdviceLookup* etc. are very different from the *Unadvised* and *NextAdvice* predicates, in that the lookup functions have a fixed interpretation and can hence simply be unfolded in any rule instances, whereas the meaning of *Unadvised* and *NextAdvice* is *defined* in (Unadv) and (NextAdv), similarly to how $\hookrightarrow$ is defined in (Meth) and (Adv).

[2]We are cheating a bit because the program is usually (at least implicitly) a part of the derivation rules, e.g., derivation rules of the format $P \vdash e_1 \hookrightarrow e_2$. We have deliberately removed the program from the rules such that we can talk about preservation of rule instances with respect to program expansion.

Assuming some module structure in the underlying language, we say that a program $P'$ is an *expansion* of $P$, if $P'$ contains $P$ but may contain additional modules. The definition of modular reasoning is as follows: *A language admits modular reasoning with respect to a set of rules, if, for all programs $P$ and $P'$ such that $P'$ is an expansion of $P$, we have $ruleinstances(P) \subseteq ruleinstances(P')$.*

The rationale behind this definition is that the set of rule instances of a program can be considered as the knowledge base which we use to reason about the program. If we investigate a subpart of a program, then the knowledge base (i.e., the set of rule instances) should only grow when we investigate bigger parts of the program, but the knowledge base should never be invalidated by considering a larger part of the program.

Let us now consider how the language definitions in Fig. 1 and 2 perform with respect to this definition. The decisive rule in Fig. 1, which prevents modular reasoning, is (Weave). For example, we may have

$$...anObj.aMeth() \hookrightarrow ...anObj.aMeth[\emptyset]()$$

for some method *aMeth* with zero parameters and object *anObj* in a program $P$, but

$$...anObj.aMeth() \hookrightarrow ...anObj.aMeth[anAdv]()$$

in an expansion $P'$ of $P$ that adds an advice *anAdv* for calls to *anObj.aMeth*(). Hence, modular reasoning (according to our definition), is not supported via this language definition.

The situation is different in Fig. 2, because there is no rule like (Weave) that needs global knowledge. To be concrete, assume a method call $anObj.aMeth[\emptyset]()$, where the body of *aMeth* just returns **this**. Then we have a rule instance of the rule schema (Meth) which has the form $\frac{Unadvised(anObj, aMeth, \emptyset)}{...anObj.aMeth[\emptyset]() \hookrightarrow ...anObj}$. This rule instance is stable w.r.t. program expansion. If we consider again $P'$ which adds advice *anAdv*, then this rule instance is still valid, but we get an additional rule instance of (SomeAdv), namely

$$\frac{true \wedge true}{\neg unadvised(anObj, aMeth, \emptyset)}$$

Hence, modular reasoning (according to our definition) is possible in the default logic version of the language semantics.

We believe that our approach also enables a form of modular verification in the sense of [14]: To determine whether an expansion of a program violates some property of the original program that holds in some extension, it is sufficient to check whether the set of assumptions $A$ in our algorithm for computing extensions is consistent with the program expansion; it is not necessary to re-examine the whole program.

## 5. PRIORITIES

If two advice apply at some joinpoint, the question arises in which order the advice are to be executed. Languages like AspectJ leave the order unspecified (or use an arbitrary order such as lexicographic order of aspect names) by default, but enable the programmer to insert precedence rules into the program. Such mechanisms are very naturally supported in default logic, and we believe that the various results in this domain (see [6, 10, 23] for an overview) could be projected back to AO languages and lead to better priority specification mechanisms.

At this point, we will only consider two simple variants of default logic with priorities: PDL and PRDL [6] [3, Chap. 8]. In PDL, the priority information is given in the form of a *strict partial order* $<$ on the set of defaults. The set of extensions of a default theory in PDL is restricted to those extensions that respect $<$, i.e., the order of default application in the algorithm in Sec. 2 is compatible with $<$.

For the purpose of modelling constructs like `declare precedence` in AspectJ, PRDL is even more appropriate, because PRDL allows to model the priority information *within* the logic, rather than as an external partial order as in PDL. In PRDL, every default $\delta_i$ has a name $d_i$. It also introduces a special symbol $\prec$ acting on default names. $d_1 \prec d_2$ can be read as "give the default with name $d_1$ priority over the default with name $d_2$". A term $d_1 \prec d_2$ in PRDL is an ordinary formula that can be used both in the background theory $W$ and in defaults $D$ of a default theory $T = (W, D)$. So, an AspectJ precedence declaration `declare predence a1, a2` can be represented by adding $d \prec d'$ to $W$ for every $d, d'$ that is the name of a rule instance of (NextAdv) for `a1` and `a2`, respectively. Of course, in PRDL the notion of extension is refined to *priority extensions*, which respect the order hints in $T$. Note that PRDL is already a much more general model than AspectJ's `declare precedence`, because ordering hints can be given inside arbitrary logic formulaes. Since they may also be given inside defaults, it is even possible to model that different extensions use different priorities! With regard to aspect priority this would mean that the priority between two advice might depend on the choosen priority order between other advice.

One step further would be to consider *dynamic priorities*, which are well-known in nonmonotonic logic [7, 5]. We believe that these mechanisms could be directly used to design new advanced priority mechanisms for AO languages.

## 6. DISCUSSION

Using default logic, it is possible to define the semantics of an AO language in a compositional way, without using weaving or other kinds of global operations. This is not only interesting from the perspective of defining the language, but also from the perspective of reasoning about programs in the language, since a language semantics also influences how we reason about programs.

Our definition of modular reasoning seems to be a bit strange in that the whole concept of interfaces, which is usually a central notion of modularty, does not show up in any way. Hence, it is not clear whether our definition really fits to what people usually associate with the term "modular reasoning". If it does not, the author is happy to take any suggestion for a better name for this property. Another potential weakness of our definition is that it is possible to build up global (or at least non-local) information during execution, e.g., a list of dynamically deployed aspects that is propagated in the ... part of our reduction rules (such as in Lämmel's approach [15]). According to our definition, such an approach would still allow modular reasoning. This brings up the question of the difference (with respect to modular reasoning) between having to have global knowledge about the program, or knowledge about dynamic parameters that influence the execution and are propagated through the execution steps (such as aspect registries, heaps, or monads).

One may also argue that our definition of modular reasoning has been carefully worded to fit to our approach, and that what one really wants is monotonicity in *the set of conclusions* from a knowledge base and not so much monotonicity in the knowledge base itself. Indeed, our default logic version is nonmonotonic in this regard: A previously existing extension based on the assumption $\mathit{unadvised}(\mathit{anObj}, \mathit{aMeth}, \emptyset)$ can be invalidated by program expansion. This is what nonmonotonic logic is about, after all.

However, we still believe there is value in our approach because now we can deal with this nonmonotonicity in a reasoning framework that has been specifically developed for that very purpose. In Sec. 5 we have already hinted at how variants of nonmonotonic logic with priorities might fertilize AO language design. We believe that this is also true for other results from nonmonotonic logic. For example, the theory of default logic gives conditions under which extensions are unique or under which conclusions are contained in all extensions of a default theory [17, 13]. There is a systematic process to deal with changing belief sets [1]. There are mechanisms to keep track of the beliefs upon which we base our conclusions [16]. With our approach, we can now project these results from default logic back into the AO language domain. Connections between logic and programming have turned out to be quite fruitful in the past (Curry-Howard isomorphism!), and we hope that this connection between AOP and nonmonotonic logic is no exception.

In this work, we have concentrated on default logic. It would probably also be possible to define our language semantics in *autoepistemic logic* [21]. Autoepistemic logic introduces an operator $L$, where $L\phi$ is interpreted as 'I believe in $\phi$'. Using this operator, our (Unadv) rule, for example, could be encoded as

$$\neg L \neg \mathit{unadvised}(o, m, \vec{a}) \rightarrow \mathit{unadvised}(o, m, \vec{a})$$

Since autoepistemic logic is intuitively based on introspection (rather than default rules), autoepistemic logic might provide another interesting reasoning framework to interpret AOP.

Another well-known approach in nonmonotonic logic is circumscription [19, 20]. We believe that circumscription could be useful to devise a model-theoretic interpretation of AOP. From the perspective of logic, most semantic accounts of AOP are proof-theoretic (including this one). Circumscription gives a model-theoretic interpretation of nonmonotonic logic by selecting minimal models from the space of models of a theory. We believe it would be possible to define a variant of our semantics where the *unadvised* and *NextAdvice* predicates are circumscribed (i.e., their meaning is minimized), rather than defining them via defaults. However, this is clearly a topic for future work.

## 7. REFERENCES

[1] C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.

[2] J. Aldrich. Open modules: Modular reasoning about advice. In *ECOOP'05*, Lecture Notes in Computer Science, pages 144–168. Springer, 2005.

[3] G. Antoniou. *Non-monotonic reasoning.* MIT Press, 1996.

[4] G. Antoniou. A tutorial on default logics. *ACM Comput. Surv.*, 31(4):337–359, 1999.

[5] G. Antoniou. Defeasible logic with dynamic priorities. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, pages 521–525. IOS Press, 2002.

[6] G. Brewka. Reasoning about priorities in default logic. In *Proceedings of the 12th national conference on Artificial intelligence (AAAI)*, pages 940–945. American Association for Artificial Intelligence, 1994.

[7] G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *J. Artif. Intell. Res. (JAIR)*, 4:19–36, 1996.

[8] C. Clifton and G. T. Leavens. Obliviousness, modular reasoning, and the behavioral subtyping analogy. In *Workshop on Software engineering Properties of Languages for Aspect Technologies (SPLAT!) at AOSD 2003.*, 2003.

[9] D. S. Dantas and D. Walker. Harmless advice. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'06)*, pages 383–396. ACM, 2006.

[10] P. M. Dung and T. C. Son. An argument-based approach to reasoning with specificity. *Artif. Intell.*, 133(1-2):35–85, 2001.

[11] R. Jagadeesan, A. Jeffrey, and J. Riely. A calculus of untyped aspect-oriented programs. In *ECOOP 2003 - Object-Oriented Programming, 17th European Conference*, pages 54–73, 2003.

[12] G. Kiczales and M. Mezini. Aspect-oriented programming and modular reasoning. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 49–58, New York, NY, USA, 2005. ACM Press.

[13] S. Kraus, D. J. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artif. Intell.*, 44(1-2):167–207, 1990.

[14] S. Krishnamurthi, K. Fisler, and M. Greenberg. Verifying aspect advice modularly. In *SIGSOFT FSE'04*, pages 137–146. ACM, 2004.

[15] R. Lämmel. A semantical approach to method-call interception. In *AOSD '02: Proceedings of the 1st international conference on Aspect-oriented software development*, pages 41–55, New York, NY, USA, 2002. ACM Press.

[16] W. Lukaszewicz. Considerations on default logic: an alternative approach. *Computational Intelligence*, 4:1–16, 1988.

[17] D. Makinson. General patterns in nonmonotonic reasoning. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*, pages 35–110, New York, NY, USA, 1994. Oxford University Press, Inc.

[18] W. Marek and M. Trusczynski. *Nonmonotonic Logic.* Springer, 1993.

[19] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[20] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[21] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.

[22] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[23] X. Zhao. Complexity of argument-based default reasoning with specificity. *AI Commun.*, 16(2):107–119, 2003.