# Aspects and Modular Reasoning in Nonmonotonic Logic

Klaus Ostermann

Darmstadt University of Technology

# Background

- Many people have noted that programs should "look like" our thought process about the problem.
  - direct mapping principle (Meyer)
  - low representational gap (Larman)
  - logical vs. physical hierarchies (Wegner)
  - ...
- However, research from the AI community on how humans think has so far had little impact on PL research

# Overview

- Fundamental insight in AI research: Humans reason in a non-monotonic way. Humans reason frequently with incomplete or changing information.
  - New knowledge may invalidate previous conclusions
- Example: Birds usually fly and Tweety is a bird $\Rightarrow$ Tweety flies.
- Later we learn that Tweety is a penguin...
- In classical logic, if $\Gamma \vdash X$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \vdash X$.
  - Not possible to express "rules of thumb" or defaults as above in classical logic.
- Nonmonotonic logic has been developed to deal with nonmonotonicity in a rigorous and controlled way.

# Hypothesis of this work

- Aspects can be interpreted as a form of nonmonotonicity
  - We can give a "default meaning" to a computational entity
  - Later (when we learn about a different concern) we can refine the meaning of this entity.
- To validate the hypothesis we perform three experiments:
  - Modeling the semantics of an AO language using nonmonotonic logic.
  - Modeling advice precedence rules with prioritized default logic.
  - Revisit the question of modular reasoning and modular verification on the basis of a semantics in default logic.

# Default Logic

- Default logic is the best-known variant of nonmonotonic logics.
- Our rule about birds can be expressed as follows:

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

- A default $\frac{\varphi:\psi_1,\ldots,\psi_n}{\chi}$ is applicable to a deductively closed set of formulae $E$, if $\varphi \in E$ and $\neg\psi_1 \notin E, \ldots, \neg\psi_n \notin E$.
- Set of conclusions from a knowledge base is in general not unique.
- Possible consistent world views from a knowledge base $T = (W, D)$ are called extensions.
- Normal defaults...

# Algorithm to compute extensions

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

# AO semantics in the style of Jagadeesan et al

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

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

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

# AO semantics in the style of Jagadeesan et al

- Semantics requires global operation that requires knowledge of the full program to compute the list of all advice that applies: $ApplicableAdvice$
- There is no direct specification of the semantics of an aspect, but just a specification of what its effect on the program is.
- Hence, the set of rule instances does not grow monotonically with the program.
- Next up: AO semantics using defaults
- To get rid of the global advice list, we re-interpret the advice list in a method call to mean the set of already executed advice.

# AO semantics using defaults

$$\frac{MethodLookup(o, m) = (\vec{x}, e) \quad unadvised(o, m, \vec{a})}{...o.m[\vec{a}](\vec{v}) \hookrightarrow ...e\left[{}^{o}/_{\mathbf{this}}, {}^{\vec{v}}/_{\vec{x}}\right]} \quad \text{(METH)}$$

$$\frac{NextAdvice(o, m, \vec{a}) = a \quad AdviceLookup(a) = (\vec{x}, e)}{...o.m[\vec{a}](\vec{v}) \hookrightarrow ...e\left[{}^{o}/_{\mathbf{this}}, {}^{\vec{v}}/_{\vec{x}}, {}^{o.m[a,\vec{a}](\vec{v})}/_{\mathsf{proceed}}\right]} \quad \text{(ADV)}$$

$$\frac{true \ : \ unadvised(o, m, \vec{a})}{unadvised(o, m, \vec{a})} \quad \text{(UNADV)}$$

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

$$\frac{a \in ApplicableAdvice(o, m) \wedge a \notin \vec{a}}{\neg unadvised(o, m, \vec{a})} \quad \text{(SOMEADV)}$$

# AO semantics using defaults

- A global list of all advice that apply at some point is never required.
- Rule instances are preserved by program expansion.
- An aspect is given a (logical) meaning independent of the program to which it applies.
- If at most one pointcut applies at any joinpoint, the two semantics agree because:
  - 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.
  - Order returned by $ApplicableAdvice$ vs. one extension for every possible execution order
- Next up: prioritized default logic to model AspectJ-like global orders and ordering hints (such as declare precedence in AspectJ) on advice.

# Prioritized Default Logic (PRDL)

- In PRDL, every default $\delta_i$ has a name $d_i$.
- ... and has a special symbol $\prec$ operating on default names.
- $d_i \prec d_j$ means $d_i$ has priority over $d_j$.
- Formulae containing $\prec$ can be used both in the background theory and in default rules.

# Algorithm to compute priority extensions

$E := Th(W); A := \emptyset; Prio := \emptyset$
while there is a default $\delta \notin A$ that is applicable to E {
  $C := \{nameof(\delta') \mid \delta' \in D, \delta' \neq \delta, \delta'$ is applicable to $E\}$
  $Prio := Prio \cup \{nameof(\delta) \prec d \mid d \in C\}$
  $E := Th(E \cup \{consequent(\delta)\}); A := A \cup \{\delta\};$
}
if $E$ is consistent with $Prio$
  then return $E$ else failure

# Modeling AspectJ-like priorities in PRDL

$$\frac{\mathit{true} \;\; \textbf{:} \;\; \mathit{defaultOrder}(\{a_1, a_2\})}{\mathit{defaultOrder}(\{a_1, a_2\})} \quad (\textsc{Default})$$

$$\frac{\mathit{defaultOrder}(\{a_1, a_2\}) \wedge (a_1 <_{\mathit{default}} a_2)}{\textsc{NextAdv}_{o,m,\vec{a},a_1} \prec \textsc{NextAdv}_{o,m,\vec{a},a_2}} \quad (\textsc{DeclDeflt})$$

# Modeling AspectJ-like priorities in PRDL

$$\frac{\mathit{true} \;\; \texttt{:} \;\; \mathit{defaultOrder}(\{a_1, a_2\})}{\mathit{defaultOrder}(\{a_1, a_2\})} \quad (\textsc{Default})$$

$$\frac{\mathit{defaultOrder}(\{a_1, a_2\}) \wedge (a_1 <_{\mathit{default}} a_2)}{\textsc{NextAdv}_{o,m,\vec{a},a_1} \prec \textsc{NextAdv}_{o,m,\vec{a},a_2}} \quad (\textsc{DeclDeflt})$$

$$\frac{\texttt{declare precedence a}_1\texttt{,a}_2 \in P}{\neg \mathit{defaultOrder}(\{a_1, a_2\})} \quad (\textsc{DeclPrec1})$$

$$\frac{\texttt{declare precedence a}_1\texttt{,a}_2 \in P \;\; \texttt{:} \;\; (\textsc{NextAdv}_{o,m,\vec{a},a_1} \prec \textsc{NextAdv}_{o,m,\vec{a},a_2})}{\textsc{NextAdv}_{o,m,\vec{a},a_1} \prec \textsc{NextAdv}_{o,m,\vec{a},a_2}} \quad (\textsc{DeclPrec2})$$

# Modeling AspectJ-like priorities in PRDL

- Again, the precedence declarations are given a compositional semantics, independent of the rest of the program.
- Semantics agrees with "classical" semantics in that there is only one unique extension that is equal to the theory of theclassical semantics.
- ...except if there are contradicting precedence declarations
  - Purpose of the justification in (DeclPrec2)...
- Higher-order (and dynamic) priority declarations can easily be modelled in PRDL.

# Modular Reasoning and Verification

- We believe that the absense of any global operations in the formal semantics can make a difference w.r.t. modular reasoning.
- But... what exactly is modular reasoning?
- From the perspective of logic, reasoning means the application of a proof calculus of a logic on a knowledge base.
- To reason about a program, we hence need a way to generate a knowledge base from a program and a proof calculus.

# Modular Reasoning and Verification

- Program $P'$ is an expansion of $P$ if $P$ is a part of $P'$.
- Definition: A language admits modular reasoning with respect to a $prog2kb$ function, if, for all programs $P$ and $P'$ such that $P'$ is an expansion of $P$, we have $prog2kb(P) \subseteq prog2kb(P')$.
- The set of rule instances of an operational semantics for some program is such a knowledge base.
- Observation: The default logic version of the semantics admits modular reasoning, the conventional semantics does not.

# Modular Reasoning and Verification

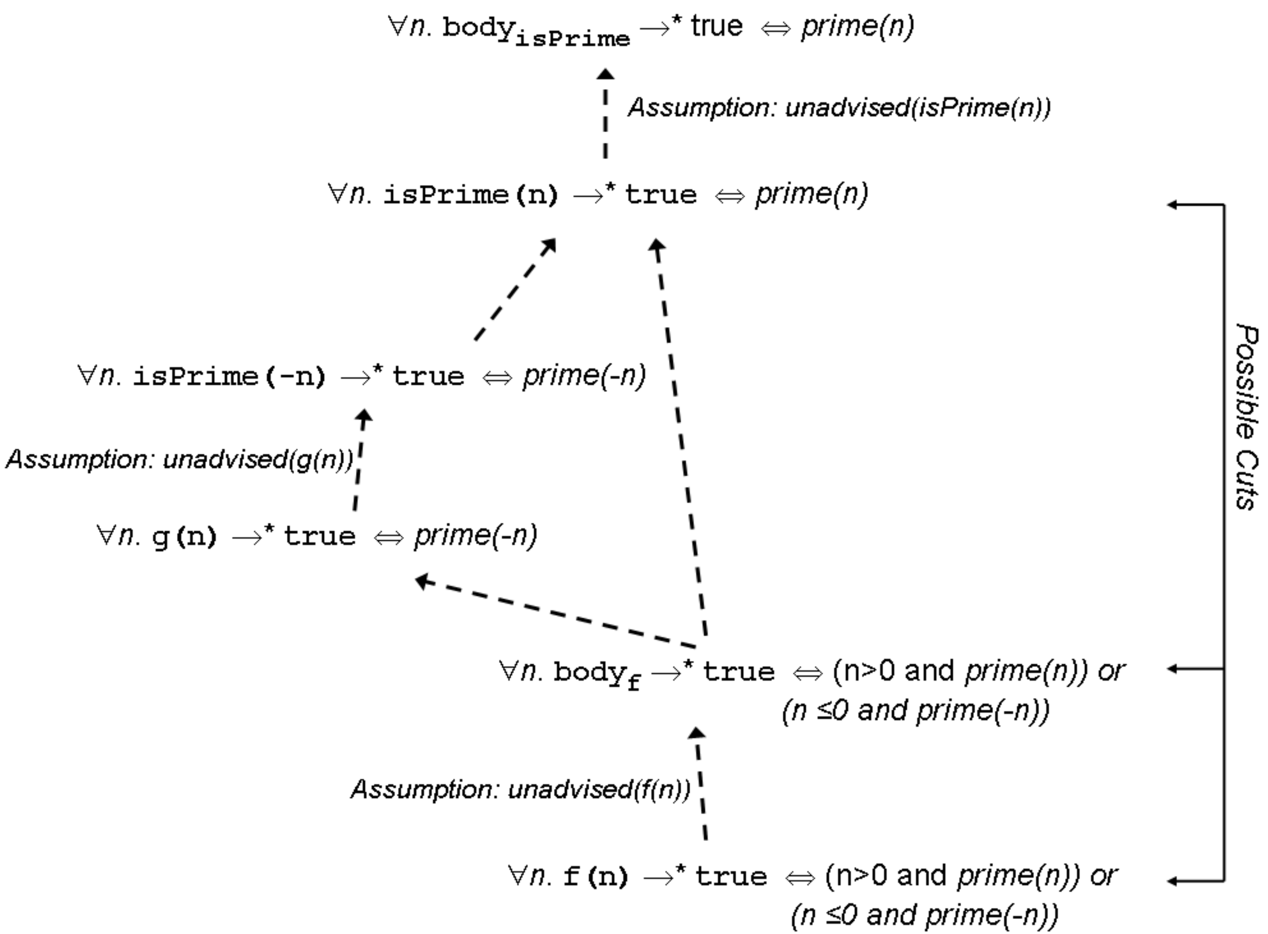
- One may argue that modular reasoning is not worth much in a nonmonotonic logic.
  - Rather than preservation of the knowledge base one would rather have preservation of the set of conclusions.
- We believe there is still value in our approach because we can now deal with the nonmonotonicity in a reasoning framework that has been specifically developed for this purpose.
- To illustrate this claim we discuss how properties of a program can be verified in a modular way.

# Example

```
bool f(int n) {
  if n<=0 then return g(n)
          else return isPrime(n);
}
bool g(int n) { return isPrime(-n); }

bool isPrime(int n) {
  if n<=1 then return false;
  for (int i=2; i<n; i++) {
    if n modulo i = 0 then return false;
  }
  return true;
}
```

# Proof of a property in default logic

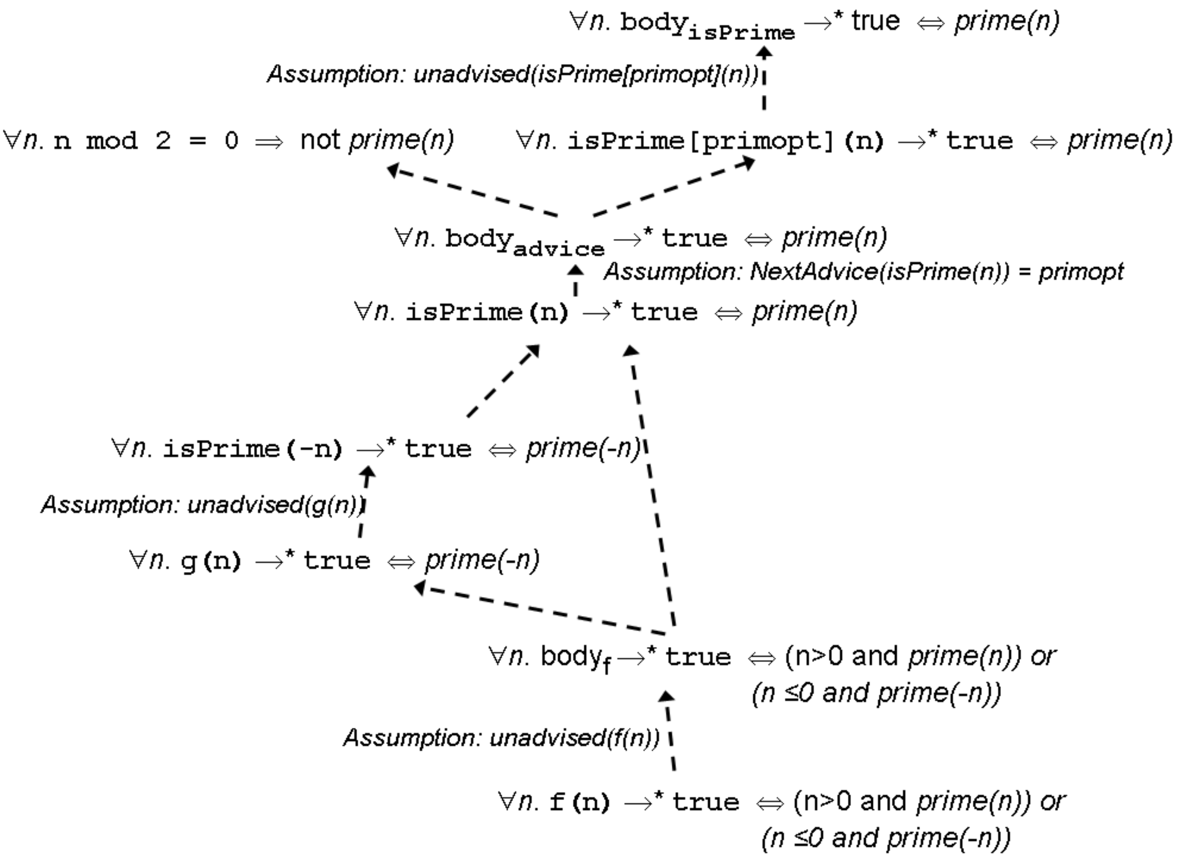


# Proof of a property in default logic

- Now consider an expansion of the program with additional advice. Is the proof $s$ (and hence property) still valid?
- Quick check: Compare whether the justification set $J(s)$ is consistent with our expansion.
- If an assumptions in $J(s)$ has been violated by the extension, however, the property may no longer hold.
- We can still try to "repair" the proof without revisiting the program.
- Example: Expansion with the following advice:

```
advice(int n) returns bool:
        around call(isPrime(n)) {
  if n % 2 = 0 then return false;
  return proceed;
}
```

# Repairing the proof



# Conclusions

- Nonmonotonic logic is a good (mental and formal) model to explain AOP.
- I hope that many results from nonmonotonic logic can be used to improve AOP
  - Semantics for AO languages
  - Advanced priority mechanisms
  - Proof theory / modular verification
- Future Work: More direct incorporation of defaults into AO languages