Methods and Rules

Serge Abiteboul* Georg Lausen† Heinz Uphoff† Emmanuel Waller*

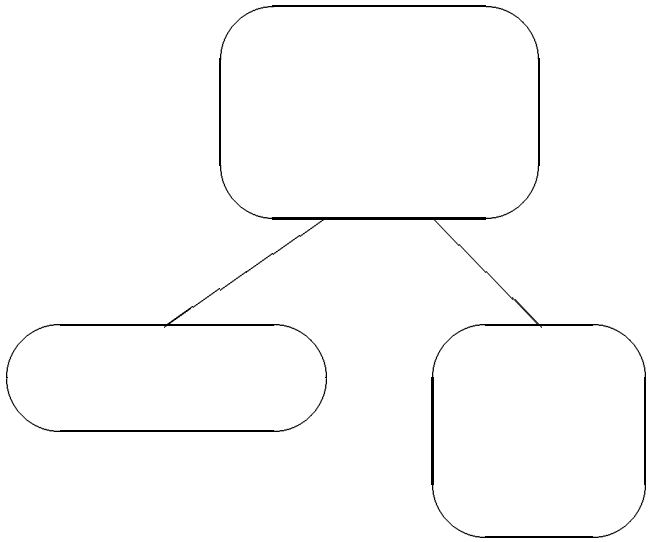
*INRIA — †University of Mannheim

Methods and Rules

Contents

- integrating methods into Datalog: Datalog^{meth}
- semantics of inheritance
- inheritance and overloading
- resolving as a general framework for overloading
- static vs. dynamic resolution
- reducing datalog^{meth} to datalog^{neg}.
- virtual classes
- flexible hierarchy
- conclusion

Integrating Methods into Datalog



OO principles

- objects classified in a hierarchy (sub- vs superclass)
- behaviour of class members represented by methods
- program structured according to classes

Datalog^{neg}

- rule language with declarative semantics
- evaluation techniques

Datalog: declarative language with a well understood semantics.

but: lack of data modeling capabilities.

Integrating oo into deductive: many problems: object identity, greater expressivness, solved eg by Datalog with Datafunctions, F-Logic.

Pragmatical approach: reducing to datalog^{neg}.

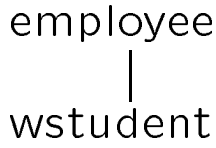
Example of a datalog^{meth} program

rules:

class : $employee$
methods: $X.salary \rightarrow Y \Longleftarrow age(X, Z),\ Y = 20 * Z.$
$X.socins \rightarrow Y \Longleftarrow X.salary \rightarrow Z,\ Y = 0.1 * Z.$

class : $wstudent$ super $employee$
methods: $X.socins \rightarrow 50 \Longleftarrow .$

hierarchy:



population:

employee	
	peter

wstudent	
	paul mary

base predicates:

age		
	peter	25
	paul	28
	mary	30

Explain Syntax: two classes, hierarchy, ,

two different method in three method definitions, valid for different classes (attachment to classes)

oids, variables, rules.

predicate rules omitted here.

Hierarchy fixed, as is set of oids.

Questions arising for a semantics of methods

1. subject of inheritance?
2. criterion for overriding?

Different meanings of the term *method:*

- a method as a function: $peter.salary \rightarrow 500$ (method atom).
- a method definition: algorithm to compute result (in a rule language: rules).

Method definitions are structured according to the class hierarchy. $\longrightarrow$ *in datalog*^{meth}, *rules are attached to classes.*

integrating inheritance into datalog: how to define a semantics? what to inherit? when not to inherit?

method is an overloaded term.

different views: structure the *world* vs. structure the *program*.

here: structure the program using the attachment (in contrast to usual F-Logic, eg).

use of attachment: restricting the domain.

Semantics of datalog^{meth}: given by a rewriting to datalog^{neg}.

First approach: the *monotonic rewriting* of the example:

$$
\begin{array}{rcll}
salary(X,Y) & \Longleftarrow & age(X,Z),\ Y = 20 * Z, & employee(X). \\
socins(X,Y) & \Longleftarrow & salary(X,Z),\ Y = 0.1 * Z, & employee(X). \\
socins(X,50) & \Longleftarrow & & wstudent(X).
\end{array}
$$

...reflects inheritance of method definitions.

$$
employee(X) \Longleftarrow wstudent(X).
$$

...reflects transitivity of class hierarchy.

Problem:

Method $socins$ is *overloaded*, i. e., multiply defined for members of class $wstudent$.

evaluate rewriting using stratified or well-founded semantics.

check for functionality

General framework for overloading: resolving

Overloading: a method call $o.m@\vec{t}$ is multiply defined

Resolving a method call: select *one* class providing a suitable method definition. Formally,

$$resolve(o.m@\vec{t}) = c.$$

$\longrightarrow$ *Resolution provides information when* not *to inherit a method definition*

Different resolution strategies presented in the sequel:

Static Resolution: based on class-membership (common in OO-programming)

Dynamic Resolution: based on domains of rules (appropriate for logic programming)

general framework to integrate and investigate different strategies.

method definition: procedure in usual languages

what is the meaning of suitable.

Different amounts of information needed to derive resolve. Different resolution strategies (i. e., $resolve$ functions) give different semantics for datalog^{meth} programs:

Static Resolution: straight adaption of OO-languages

According to this strategy, the resolved class only depends on the class of the receiver object. Inheritance is deferred if a more specific method definition is present.

Definition Given a method symbol m and a class c,

$$resolve^{static}(m, c)$$

gives the minimal superclass where a method definition is present. □

Definition Given a datalog^{meth} program Δ, a method definition $P_{m,c}$ is the set of rules attached to a class c, defining a method symbol m. □

Idea: exclude members of (redefining) subclasses from rule application.

only class information needed.

presence of method definitions: syntactical notion (compile time)

The *static rewriting* of the example:

$$
\begin{array}{lcll}
salary(X,Y) & \Longleftarrow & age(X,Z),\ Y = 20 * Z, & employee(X). \\
socins(X,Y) & \Longleftarrow & salary(X,Z),\ Y = 0.1 * Z, & employee(X), \neg wstudent(X). \\
socins(X,50) & \Longleftarrow & & wstudent(X).
\end{array}
$$

...represents blocking inheritance for redefining subclasses

Correspondence: rewriting — resolution

Definition A rewriting *respects* a resolution function, if for a (canonical) model M resulting from this rewriting the following holds: A method-atom $o.m@\vec{t} \rightarrow s$ is in M iff there is an applicable method definition $P_{m,c}$ providing this method atom and this method definition is selected according to the resolution function. □

Theorem The static rewriting of a datalog^{meth} program respects the static resolution function. □

meaning of *canonical* model!

intuitive meaning of theorem: rewriting gives the semantics of static resolution.

Limitations of static resolution/rewriting

- usually, method definitions in OO-languages are *total* definitions.
- rules are *partial* definitions.
- Static resolution is defined on a class level, implicitly assuming total methods.

Example (modified):

class $wstudent$
methods: $X.socins \rightarrow 50 \Leftarrow\!= X.salary \rightarrow S, S \leq 500.$

The domain of the method defined by this rule is in general a proper subset of the class extension.

$\longrightarrow$ the method $socins$ remains undefined for members of class $wstudent$ earning much money.

Dynamic Resolution: semantic resolution

static: resolution on class level, based on *presence* of method definitions
$\longrightarrow$ syntactic notion.

dynamic: resolution on object level, based on *applicability* of method definitions
$\longrightarrow$ semantic notion.

Definition Given a method call $o.m@\vec{t}$ in an interpretation I and a class c,

$$resolve^{dyn}(o.m@\vec{t}, I) = c$$

gives the minimal superclass where a method definition is applicable. □

The *dynamic rewriting* of the (modified) example:

$$
\begin{array}{rcll}
employee(X) & \Longleftarrow & wstudent(X). & \\
salary(X,Y) & \Longleftarrow & age(X,A), Y = 20 * A, & employee(X). \\
socins(X,Y) & \Longleftarrow & salary(X,S), Y = 0.1 * S, & employee(X), \\
 & & & \neg appl_{wstudent,socins}(X). \\
appl_{wstudent,socins}(X), & & & \\
socins(X,50) & \Longleftarrow & salary(X,S), S \leq 500, & wstudent(X).
\end{array}
$$

...applicability of rules integrated into the program

Correspondence: dynamic rewriting — dynamic resolution

Theorem The dynamic rewriting of a datalog^{meth} program respects the dynamic resolution function. □

Extension: virtual classes

Classes populated at runtime according to properties are *virtual classes* (Cf. views).

Example of a datalog^{meth} program with virtual classes:

class : $poorstudent$ super $wstudent$
instances: $poorstudent(X) \Longleftarrow X.salary \rightarrow S, S \leq 500.$

Theorem Datalog^{meth} with virtual classes and static inheritance is as expressive as datalog^{meth} with dynamic inheritance. □

Idea: introduce classes representing applicability of rules ⇒ rules are *total* definitions in these classes.

Extension: flexible class hierarchy

Classes are organized at runtime, i. e., the hierarchy is defined intensionally.

Example of a datalog^{meth,flex} program

class :	$items$	
	methods:	$X.price \rightarrow Y \Longleftarrow r(X,Y).$
class :	$bigItem$	super $items$
class :	$cheapItem$	super $items$
	methods:	$X.price \rightarrow Y \Longleftarrow s(X,Y).$
hierarchy :	$bigItem \prec cheapItem \Longleftarrow rent(X), X > 2000.$	

dynamic rewriting for programs with flexible hierarchy:

$$
\begin{array}{rcl}
bigItem \prec item. & & cheapItem \prec item. \\
bigItem \prec cheapItem & \Longleftarrow & rent(X), X > 2000. \\
X \prec Z & \Longleftarrow & X \prec Y, Y \prec Z. \\
in(O, C') & \Longleftarrow & in(O, C), C \prec C'. \\
price_{possible}(item, X, Y) & \Longleftarrow & r(X, Y),\ in(X, item). \\
price_{possible}(cheapItem, X, Y) & \Longleftarrow & s(X, Y),\ in(X, cheapItem). \\
price(X, Y) & \Longleftarrow & price_{possible}(C, X, Y),\ \neg price_{overwrite}(C, X). \\
price_{overwrite}(C1, X) & \Longleftarrow & price_{possible}(C1, X, R1),\ C2 \prec C1, \\
 & & price_{possible}(C2, X, R2),\ \neg C1 \prec C2.
\end{array}
$$

...reasoning about the hierarchy integrated into the program.

Theorem The dynamic rewriting of a datalog^{meth,flex} program respects the dynamic resolution function. □

Conclusion

- Integration of object oriented ideas into datalog
- general framework for inheritance: resolution
- putting down to semantics of datalog^{neg}
- different resolution strategies:
  - static inheritance: syntactic strategy
  - dynamic inheritance: semantic strategy
- class population defined intensionally
- class hierarchy defined intensionally

some junk...

Modeling Methods in Datalog:

predicates: $salary(john, 1000)$

methods: $john.salary \rightarrow 1000$

- methods impose functionality constraint on possible worlds,
- methods are defined in classes, serving as the domain for method application,
- method definitions (i.e. algorithms) are inherited to members of subclasses
- multiple definitions for objects have to be *resolved*

We want to introduce methods into datalog.

Difference between predicates-methods

Domain is given as the set of class members

toy-language

Semantics of a datalog^{meth} program $\triangle$

- provide a rewriting into datalog^{neg} (capturing idea of inheritance)
- evaluate using standard datalog^{neg} techniques (stratified/well-founded negation)
- check functionality constraint

AI: structural inheritance

rewrite building blocks in the intuitive way: method atoms ...