RULE MODULE INHERITANCE WITH MODIFICATION RESTRICTIONS

Felix Burgstaller
felix.burgstaller@jku.at

Bernd Neumayr, Emanuel Sallinger, Michael Schrefl

JOHANNES KEPLER UNIVERSITY LINZ

UNIVERSITY OF OXFORD
DEPARTMENT OF COMPUTER SCIENCE

UNDERLYING RULE LANGUAGE - DATALOG± / VADALOG

- Datalog±
  - Simple yet powerful
  - Extended Datalog

+	-
Existentially quantified variables	Restrictions to be decidable
Negative constraints	
Equality-generating dependencies	

- Encompasses
  - Full Datalog
  - SPARQL under OWL 2 QL entailment
- Vadalog: Datalog± implementation with many extensions

MOTIVATION FOR RULE MODULES

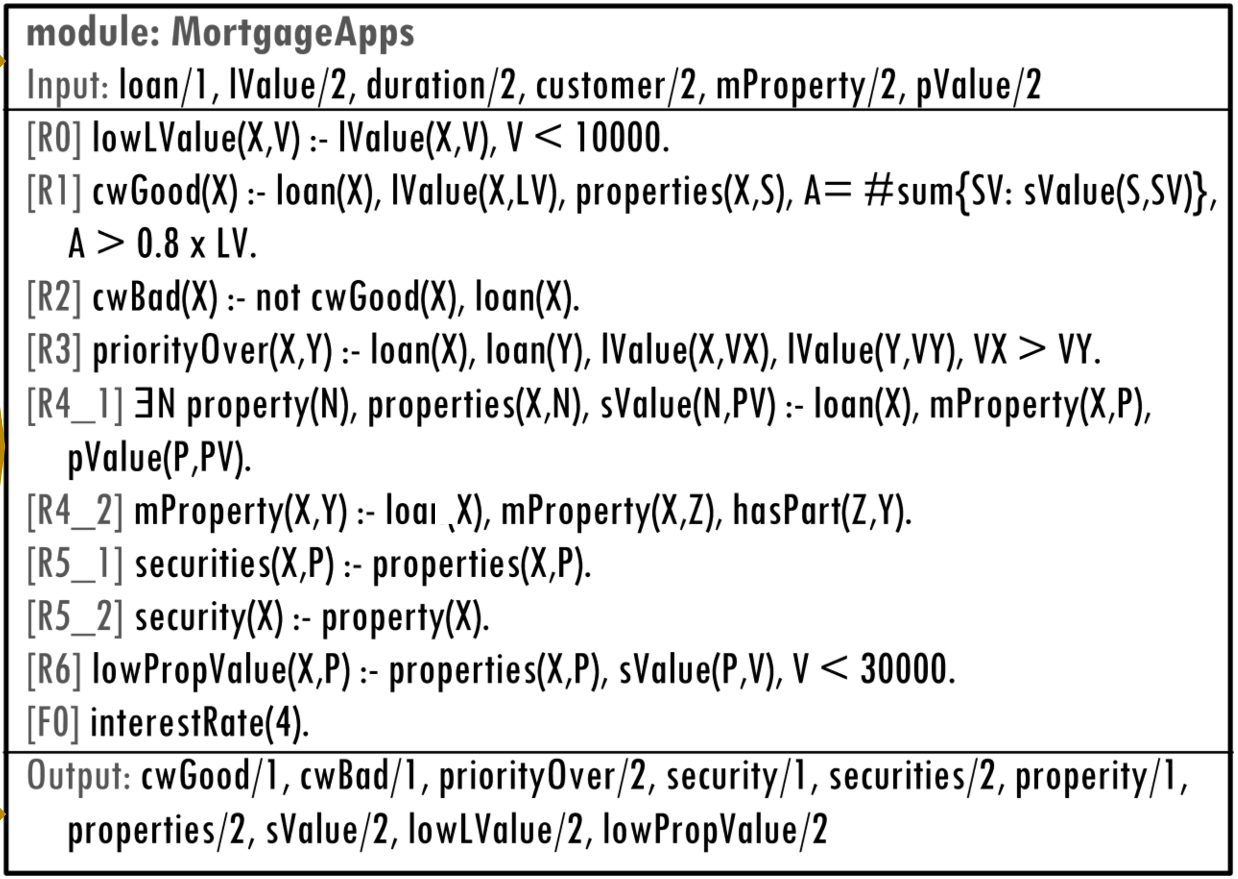
- Increasing number and complexity of rules
- Maintenance and adaption key challenges
- Separate rule-base knowledge from application code
- Clear interfaces are vital

RULE MODULES

Module Structure

Determining *Module Behavior*, i.e., derived facts in output for given input

Module Structure



```
module: MortgageApps
Input: loan/1, lValue/2, duration/2, customer/2, mProperty/2, pValue/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), properties(X,S), A= #sum{SV: sValue(S,SV)},
    A > 0.8 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[R4_1] ∃N property(N), properties(X,N), sValue(N,PV) :- loan(X), mProperty(X,P),
    pValue(P,PV).
[R4_2] mProperty(X,Y) :- loan(X), mProperty(X,Z), hasPart(Z,Y).
[R5_1] securities(X,P) :- properties(X,P).
[R5_2] security(X) :- property(X).
[R6] lowPropValue(X,P) :- properties(X,P), sValue(P,V), V < 30000.
[F0] interestRate(4).
Output: cwGood/1, cwBad/1, priorityOver/2, security/1, securities/2, properity/1,
    properties/2, sValue/2, lowLValue/2, lowPropValue/2
```

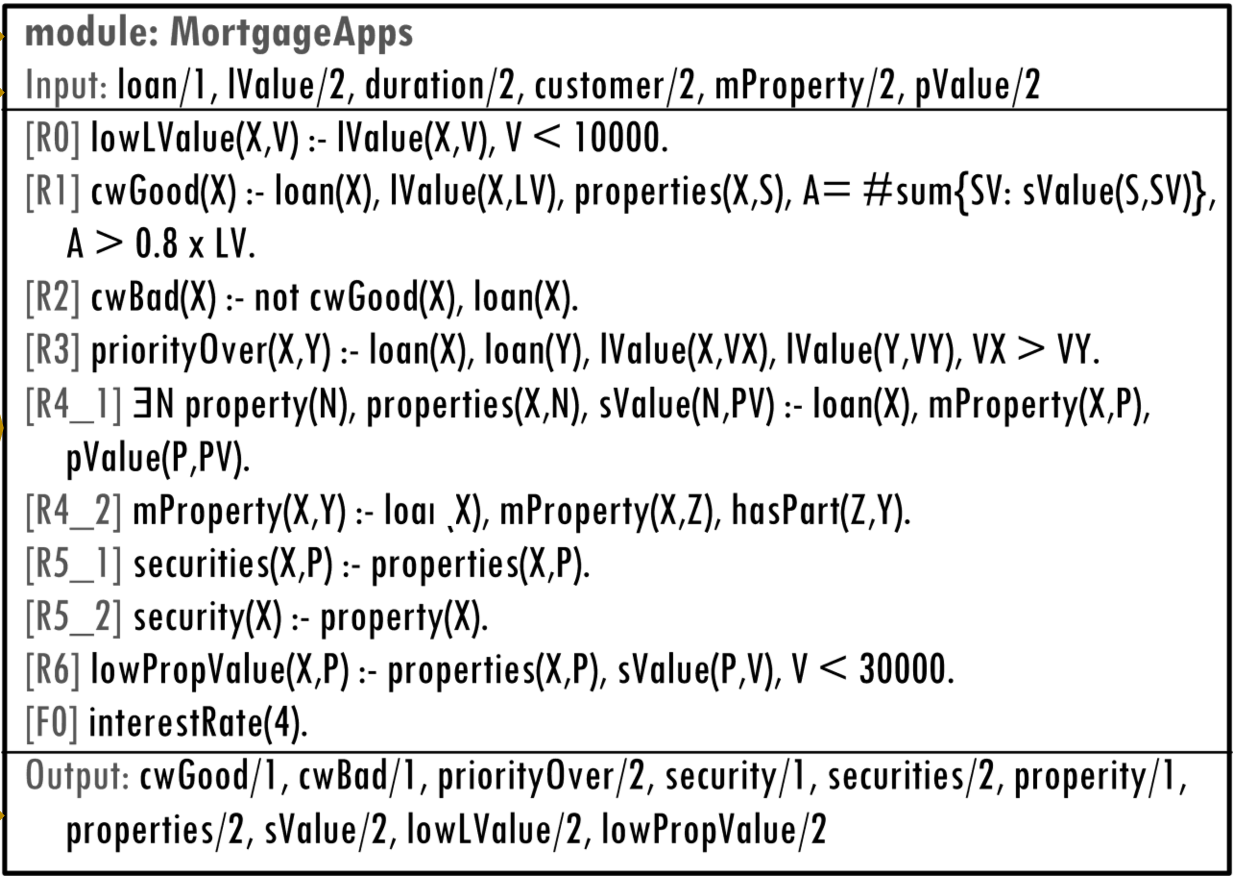
RULE MODULES

Module Name

Input Interface

Statements
- Rules
- Facts

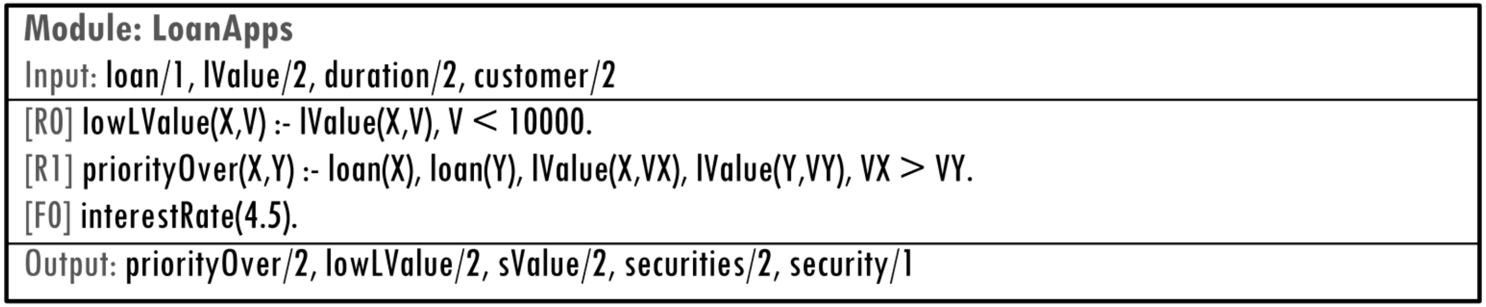
Output Interface



```
module: MortgageApps
Input: loan/1, lValue/2, duration/2, customer/2, mProperty/2, pValue/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), properties(X,S), A= #sum{SV: sValue(S,SV)},
    A > 0.8 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[R4_1] ∃N property(N), properties(X,N), sValue(N,PV) :- loan(X), mProperty(X,P),
    pValue(P,PV).
[R4_2] mProperty(X,Y) :- loan(X), mProperty(X,Z), hasPart(Z,Y).
[R5_1] securities(X,P) :- properties(X,P).
[R5_2] security(X) :- property(X).
[R6] lowPropValue(X,P) :- properties(X,P), sValue(P,V), V < 30000.
[F0] interestRate(4).
Output: cwGood/1, cwBad/1, priorityOver/2, security/1, securities/2, properity/1,
    properties/2, sValue/2, lowLValue/2, lowPropValue/2
```

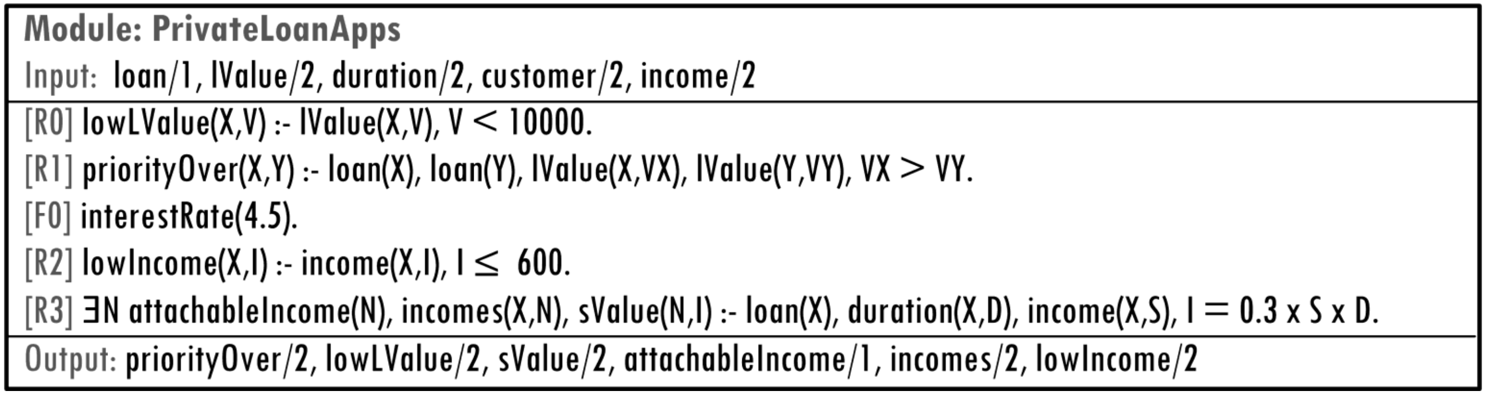
MOTIVATION FOR RULE MODULE INHERITANCE

- Module *LoanApps* describes rules and facts applying to all loan applications regardless the specific loan type
- Without inheritance, restating is necessary in *PrivateLoanApps*



Module: LoanApps

```
Input: loan/1, lValue/2, duration/2, customer/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1
```



Module: PrivateLoanApps

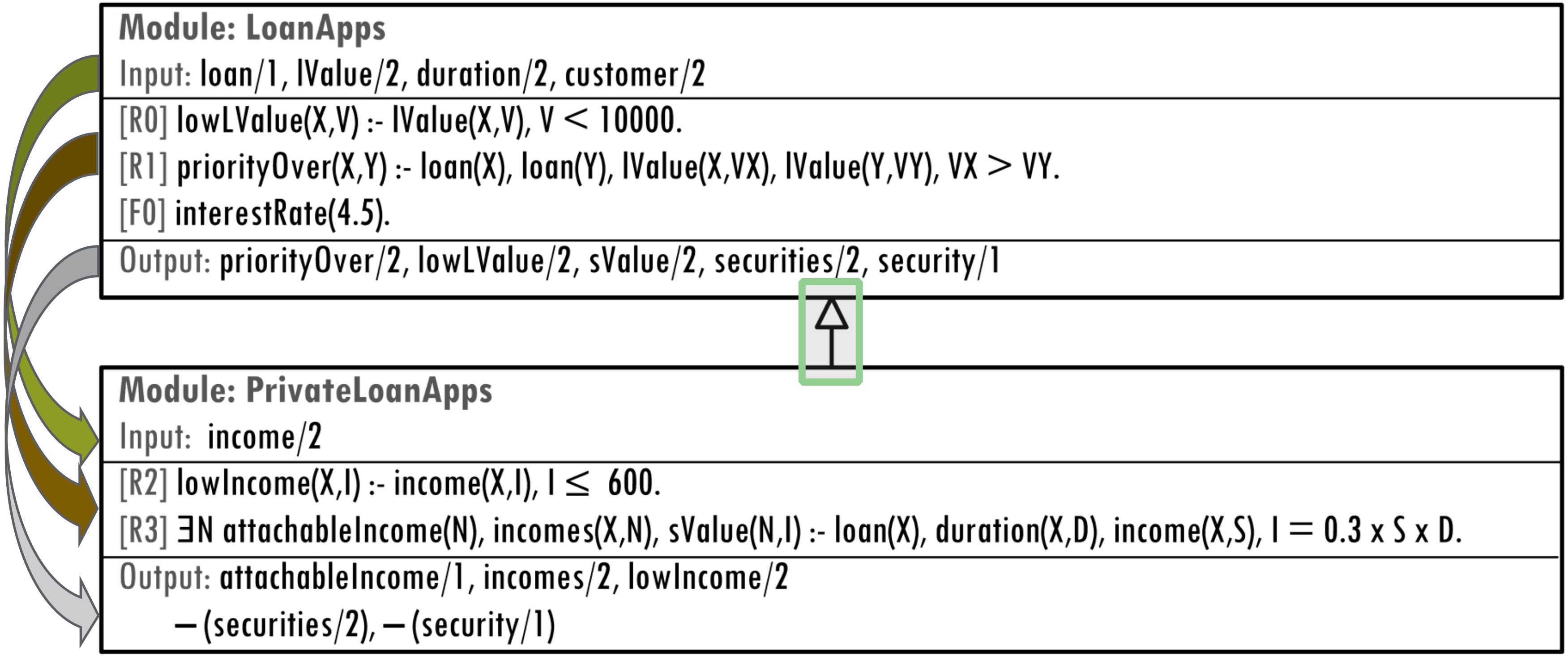
```
Input:  loan/1, lValue/2, duration/2, customer/2, income/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
[R2] lowIncome(X,I) :- income(X,I), I ≤  600.
[R3] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
Output: priorityOver/2, lowLValue/2, sValue/2, attachableIncome/1, incomes/2, lowIncome/2
```

MOTIVATION FOR RULE MODULE INHERITANCE

- Reuse of existing rules (reduce redundancy)
- Easing maintenance
- Extract common rules, facts, and interface predicates from modules
- Adaption of modules by minor modifications
- Enables more sophisticated rule organization principles

RULE MODULE INHERITANCE

- Single downward inheritance of structure and behavior



Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.

[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.

[F0] interestRate(4.5).

Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

Module: PrivateLoanApps

Input: income/2

[R2] lowIncome(X,I) :- income(X,I), I ≤ 600.

[R3] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.

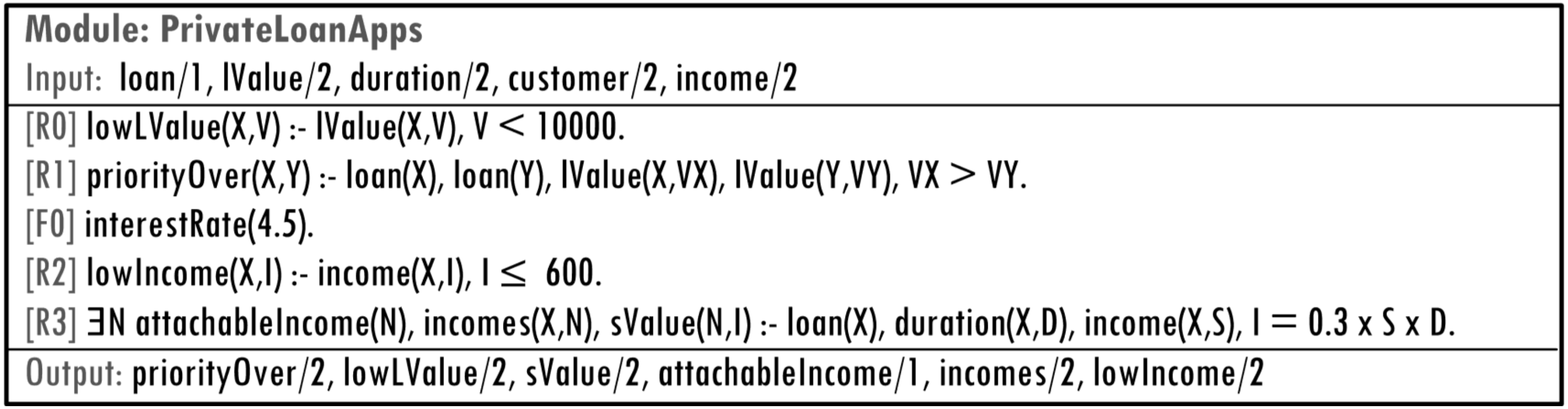
Output: attachableIncome/1, incomes/2, lowIncome/2

— (securities/2), — (security/1)

RULE MODULE INHERITANCE

- Single downward inheritance of structure and behavior

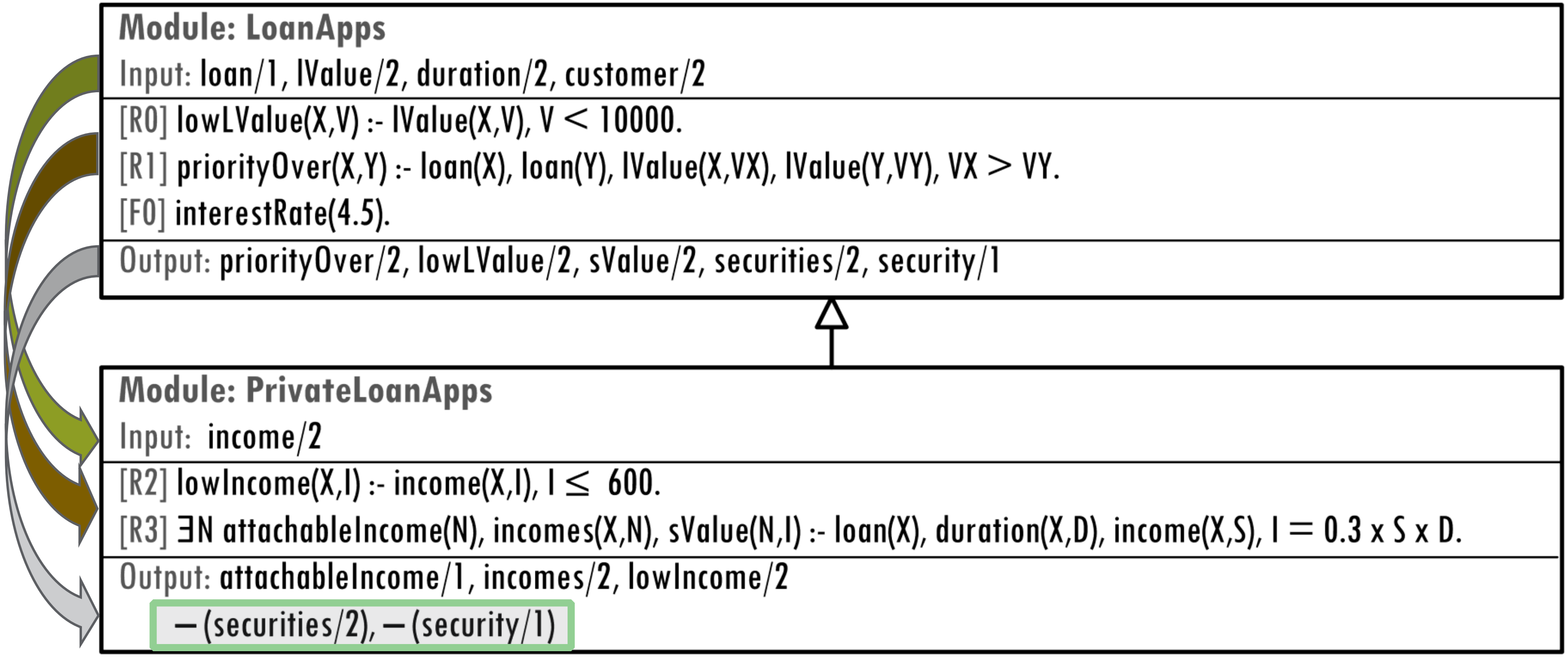
Module *PrivateLoanApps* with inheritance resolved



```
Module: PrivateLoanApps
Input:  loan/1, lValue/2, duration/2, customer/2, income/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
[R2] lowIncome(X,I) :- income(X,I), I ≤  600.
[R3] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
Output: priorityOver/2, lowLValue/2, sValue/2, attachableIncome/1, incomes/2, lowIncome/2
```

RULE MODULE INHERITANCE

- Single downward inheritance of structure and behavior
- Inherited rule module structure and behavior may be modified



Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.

[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.

[F0] interestRate(4.5).

Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

Module: PrivateLoanApps

Input: income/2

[R2] lowIncome(X,I) :- income(X,I), I ≤ 600.

[R3] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.

Output: attachableIncome/1, incomes/2, lowIncome/2

− (securities/2), − (security/1)

ABSTRACT PREDICATES AND ABSTRACT RULE MODULES

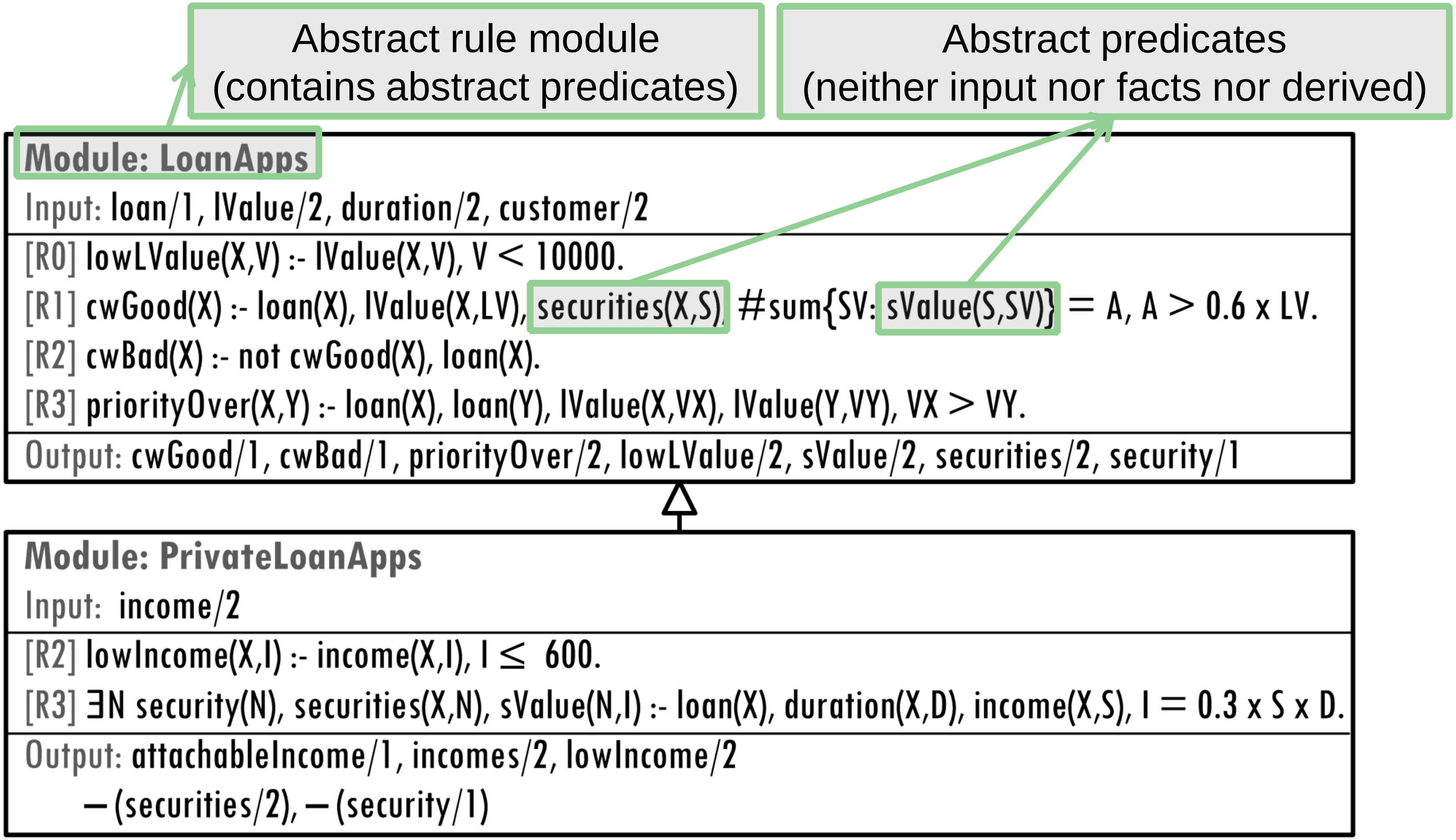
- Abstract in OO: signature defined but not implemented

- *Concrete predicate*:
  - input predicate
  - predicate with asserted facts
  - head of derivation rule(s) with only concrete predicates in body

- *Abstract predicate*: predicate not concrete

- *Abstract module*: module containing abstract predicates

ABSTRACT PREDICATES AND ABSTRACT RULE MODULES



Abstract rule module
(contains abstract predicates)

Abstract predicates
(neither input nor facts nor derived)

Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] cwGood(X) :- loan(X), lValue(X,LV), securities(X,S), #sum{SV: sValue(S,SV)} = A, A > 0.6 x LV.
[R2] cwBad(X) :- not cwGood(X), loan(X).
[R3] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.

Output: cwGood/1, cwBad/1, priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

Module: PrivateLoanApps

Input: income/2

[R2] lowIncome(X,I) :- income(X,I), I ≤ 600.
[R3] ∃N security(N), securities(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.

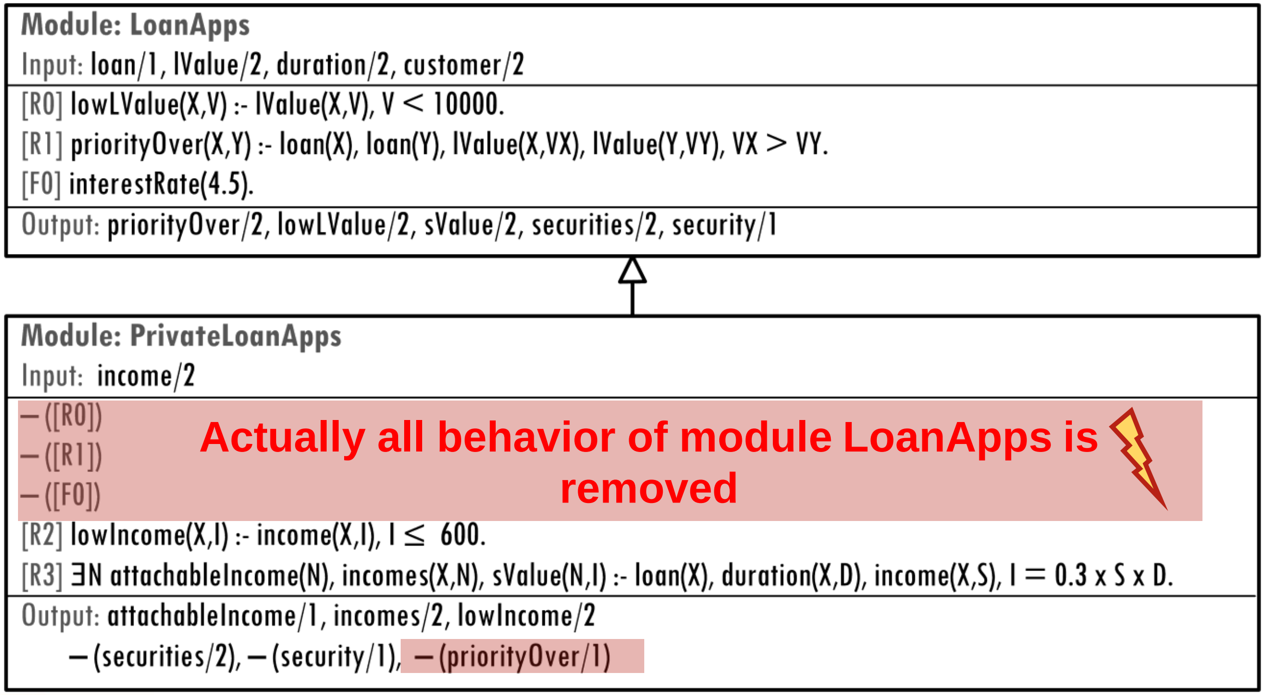
Output: attachableIncome/1, incomes/2, lowIncome/2
−(securities/2), −(security/1)

MOTIVATION FOR MODIFICATION RESTRICTIONS

- Restrict modifications allowed in child rule modules
- Vital to represent organizational constraints
- Considering parent module and restrictions gives a good overview

MOTIVATION FOR MODIFICATION RESTRICTIONS

- Restrict modifications allowed in child rule modules
- Vital to represent organizational constraints
- Considering parent module and restrictions gives a good overview



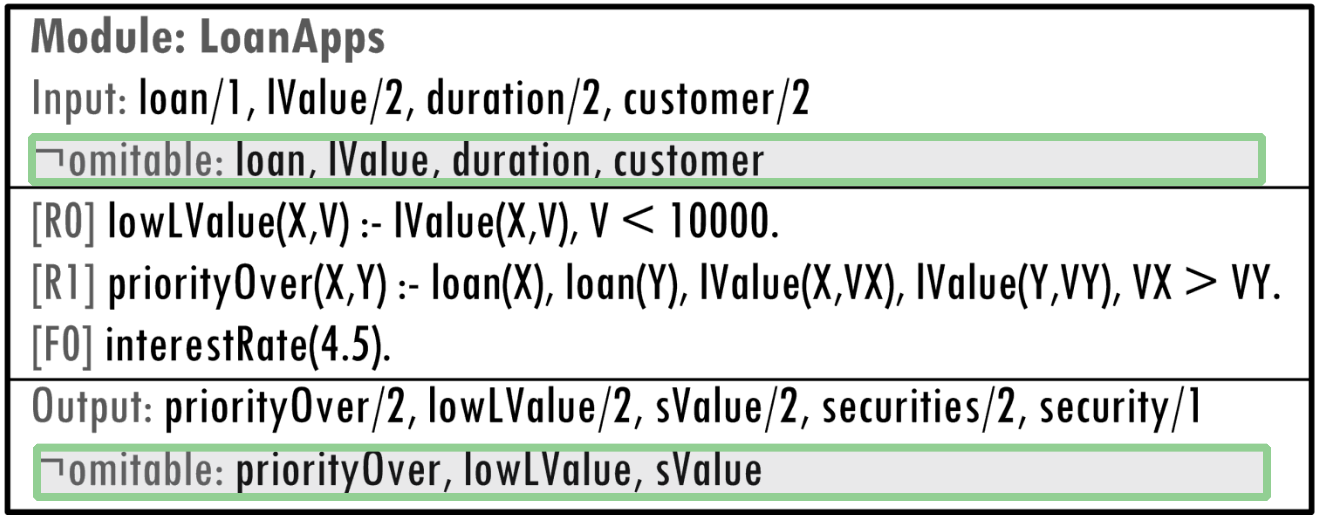
Module: LoanApps
Input: loan/1, lValue/2, duration/2, customer/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

Module: PrivateLoanApps
Input: income/2
−([R0])
−([R1])
−([F0])
[R2] lowIncome(X,I) :- income(X,I), I ≤ 600.
[R3] ∃N attachableIncome(N), incomes(X,N), sValue(N,I) :- loan(X), duration(X,D), income(X,S), I = 0.3 x S x D.
Output: attachableIncome/1, incomes/2, lowIncome/2
−(securities/2), −(security/1), −(priorityOver/1)

Actually all behavior of module LoanApps is removed

STRUCTURAL MODIFICATION RESTRICTIONS

- Consider rule module interfaces
  - □ ¬extensible input
  - □ ¬extensible output
  - □ ¬omitable
- Conformance check
  - □ Simple comparison of interface predicate sets



Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

¬omitable: loan, lValue, duration, customer

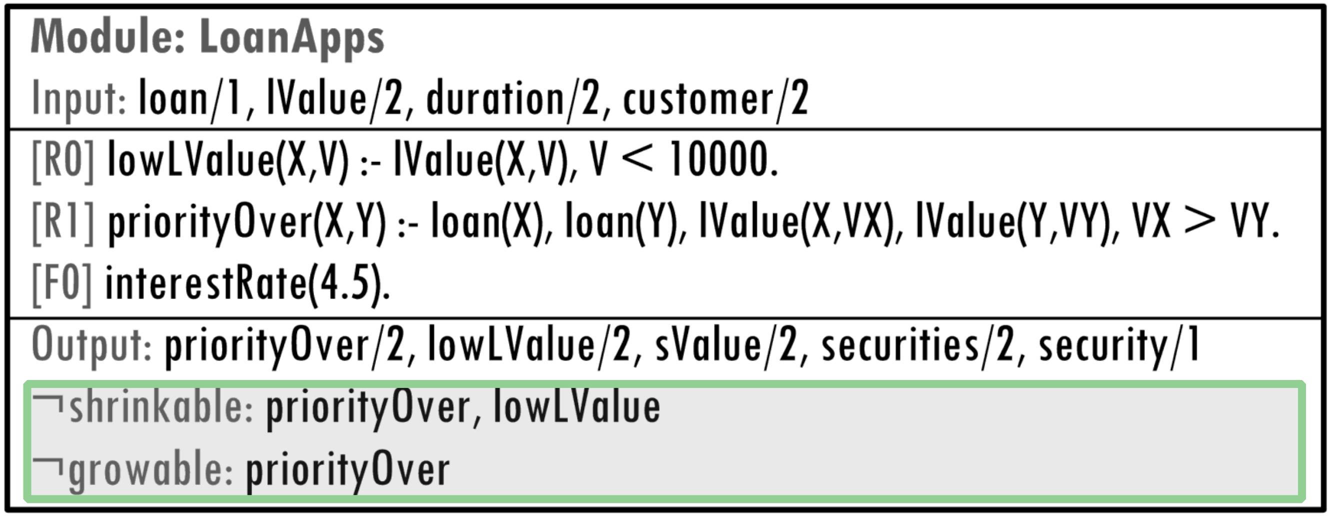
```
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
```

Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

¬omitable: priorityOver, lowLValue, sValue

BEHAVIORAL MODIFICATION RESTRICTIONS

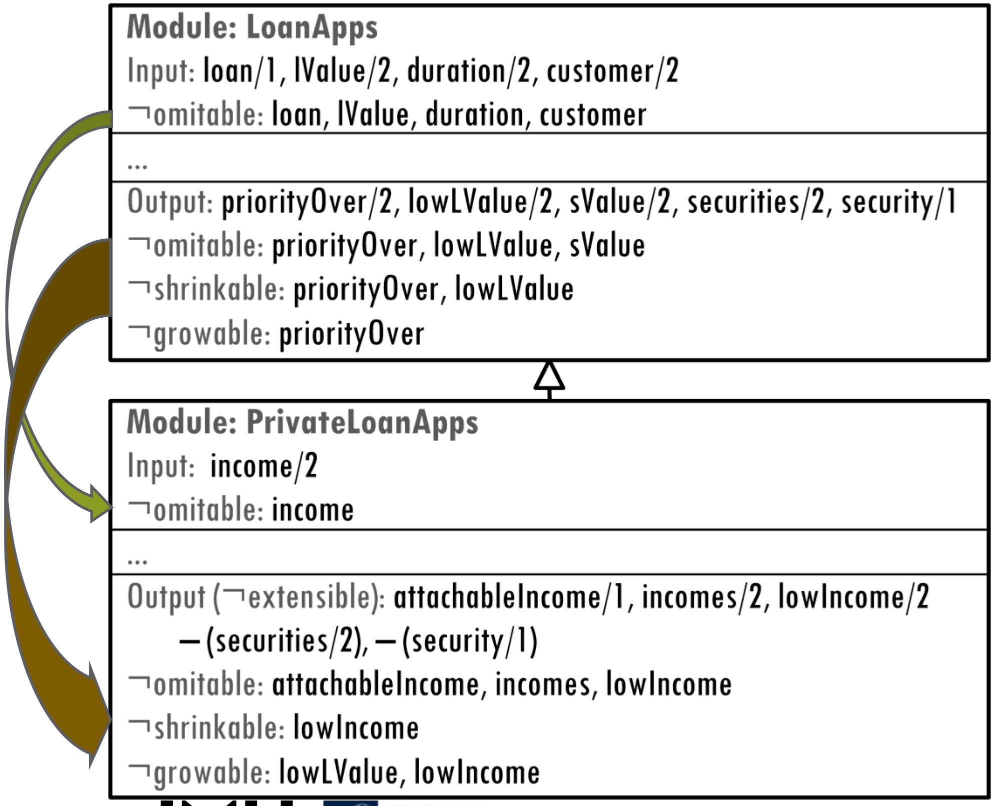
- Consider rule module behavior (derived facts for output predicates)
  - □ Prohibit additional facts for output predicate (¬growable)
  - □ Prohibit omission of facts for output predicate (¬shrinkable)
- Conformance check
  - □ Detecting modification (static, dynamic, manual)
  - □ Compare detected modification with restrictions



```
Module: LoanApps
Input: loan/1, lValue/2, duration/2, customer/2
[R0] lowLValue(X,V) :- lValue(X,V), V < 10000.
[R1] priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX), lValue(Y,VY), VX > VY.
[F0] interestRate(4.5).
Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1
¬shrinkable: priorityOver, lowLValue
¬growable: priorityOver
```

MODIFICATION RESTRICTIONS

- Modification Restrictions are inherited
- Imposed restrictions cannot be revoked



Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

¬omitable: loan, lValue, duration, customer

...

Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

¬omitable: priorityOver, lowLValue, sValue

¬shrinkable: priorityOver, lowLValue

¬growable: priorityOver

Module: PrivateLoanApps

Input: income/2

¬omitable: income

...

Output (¬extensible): attachableIncome/1, incomes/2, lowIncome/2
— (securities/2), — (security/1)

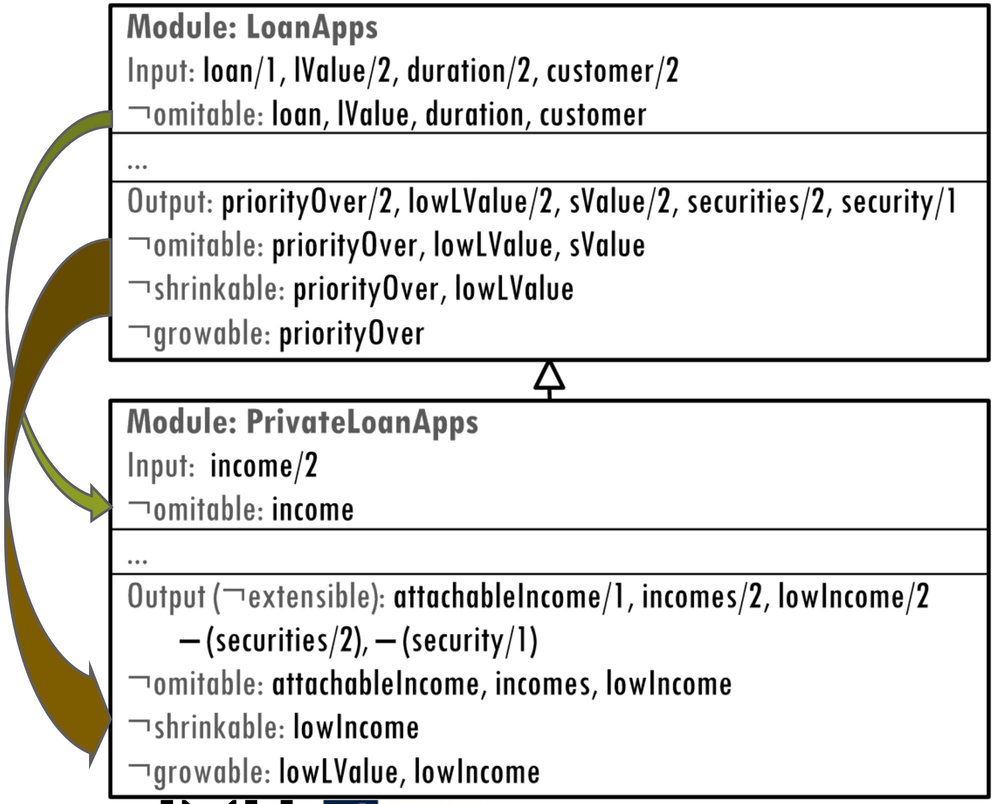
¬omitable: attachableIncome, incomes, lowIncome

¬shrinkable: lowIncome

¬growable: lowLValue, lowIncome

MODIFICATION RESTRICTIONS

- Modification Restrictions are inherited
- Imposed restrictions cannot be revoked



Module: LoanApps

Input: loan/1, lValue/2, duration/2, customer/2

¬omitable: loan, lValue, duration, customer

...

Output: priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1

¬omitable: priorityOver, lowLValue, sValue

¬shrinkable: priorityOver, lowLValue

¬growable: priorityOver

Module: PrivateLoanApps

Input: income/2

¬omitable: income

...

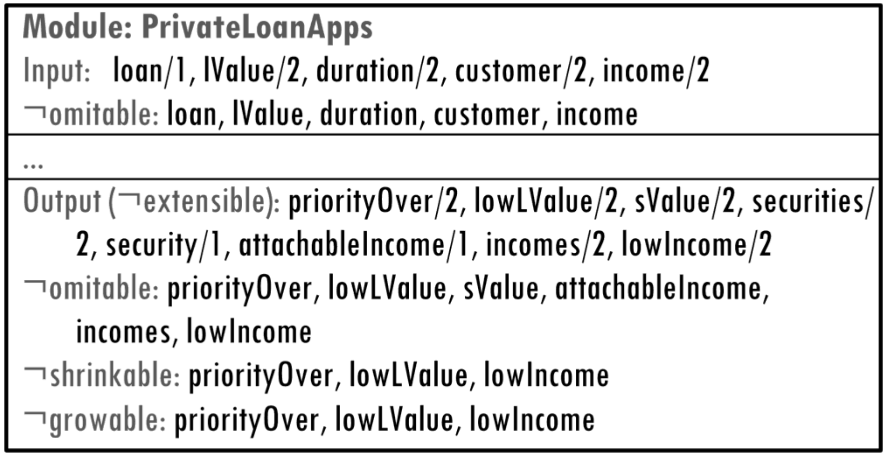
Output (¬extensible): attachableIncome/1, incomes/2, lowIncome/2 −(securities/2), −(security/1)

¬omitable: attachableIncome, incomes, lowIncome

¬shrinkable: lowIncome

¬growable: lowLValue, lowIncome

PrivateLoanApps with inheritance resolved



Module: PrivateLoanApps

Input: loan/1, lValue/2, duration/2, customer/2, income/2

¬omitable: loan, lValue, duration, customer, income

...

Output (¬extensible): priorityOver/2, lowLValue/2, sValue/2, securities/2, security/1, attachableIncome/1, incomes/2, lowIncome/2

¬omitable: priorityOver, lowLValue, sValue, attachableIncome, incomes, lowIncome

¬shrinkable: priorityOver, lowLValue, lowIncome

¬growable: priorityOver, lowLValue, lowIncome

CONCLUSION

- Investigated inheritance of rule modules to
  - Increase reuse of rules and facts
  - Simplify adaptation
  - Ease maintenance
- Investigated modification restrictions to
  - Restrain modifications to rule module structure and behavior
  - Represent organizational constraints
  - Ease and simplify obtaining an overview of the rule base
- Potential Application Areas
  - Business rule systems
  - Knowledge graph management
  - Web data extraction

JOHANNES KEPLER
UNIVERSITY LINZ

UNIVERSITY OF
OXFORD

DEPARTMENT OF
COMPUTER
SCIENCE

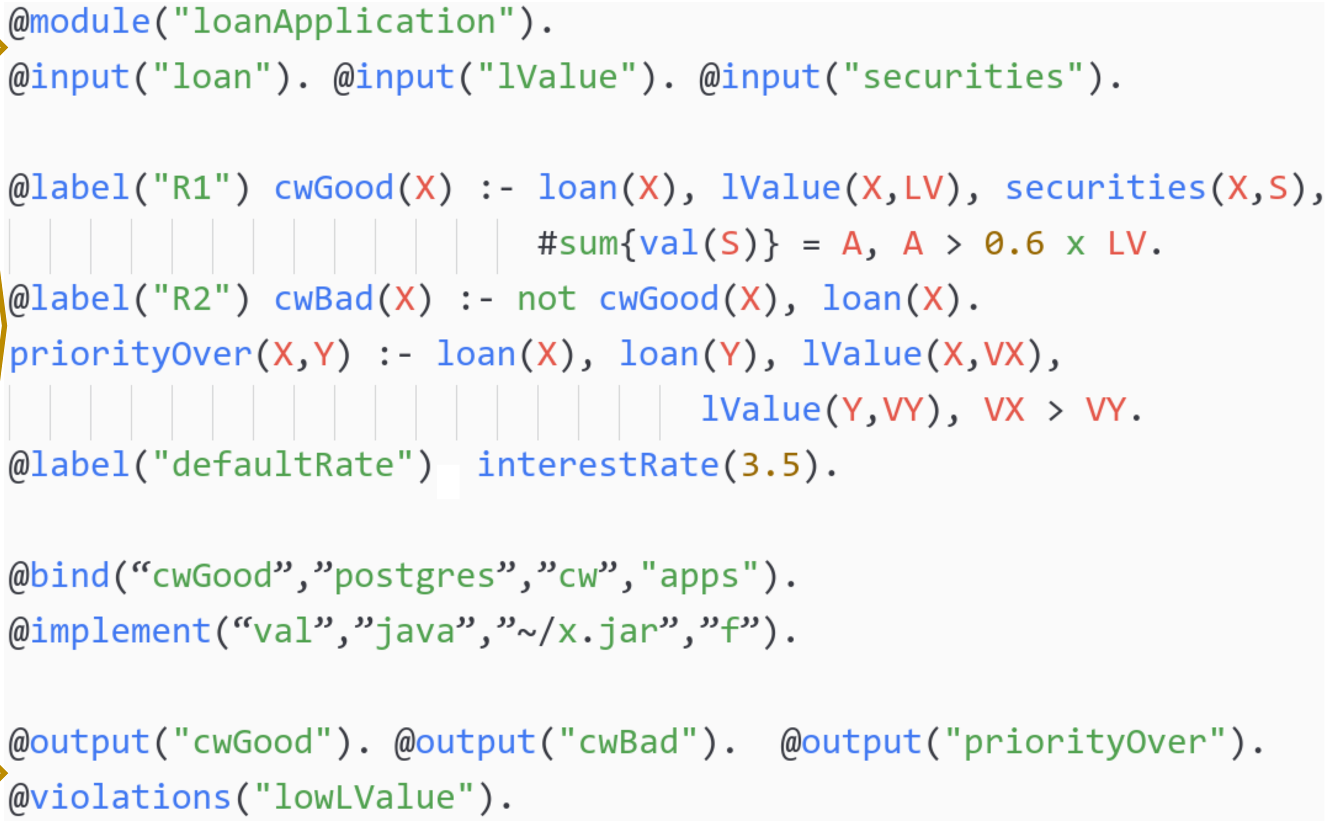
BUSINESS RULE MODULES

Module Structure

Determining Module Behavior, i.e.,

derived facts in output for given input

Module Structure



```
@module("loanApplication").
@input("loan"). @input("lValue"). @input("securities").

@label("R1") cwGood(X) :- loan(X), lValue(X,LV), securities(X,S),
                          #sum{val(S)} = A, A > 0.6 x LV.
@label("R2") cwBad(X) :- not cwGood(X), loan(X).
priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX),
                                  lValue(Y,VY), VX > VY.
@label("defaultRate")  interestRate(3.5).

@bind(“cwGood”,”postgres”,”cw”,"apps").
@implement(“val”,”java”,”~/x.jar”,”f”).

@output("cwGood"). @output("cwBad").  @output("priorityOver").
@violations("lowLValue").
```

BUSINESS RULE MODULES

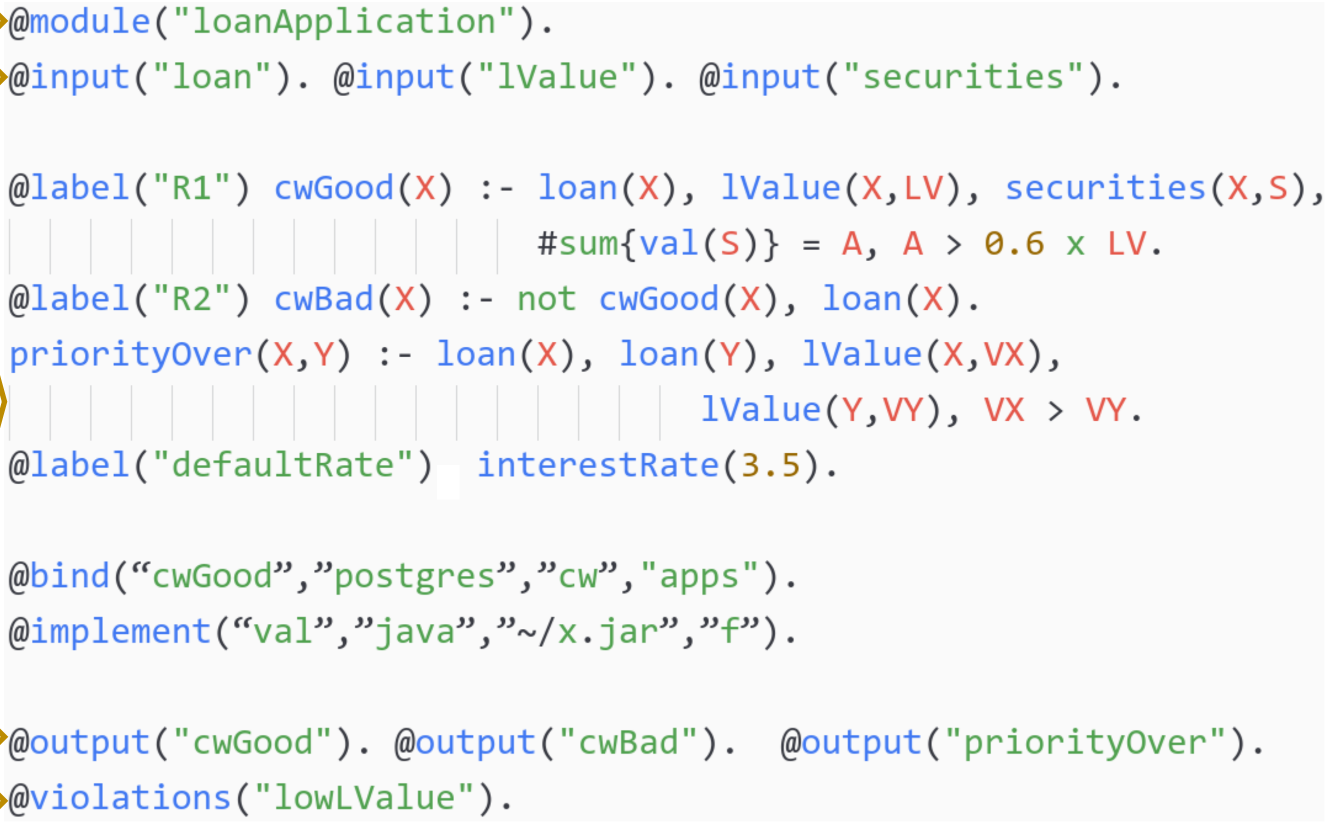
Module Name

Input Interface

Statements
- Rules
- Facts
- Annotations

Output Interface

Violation Interface



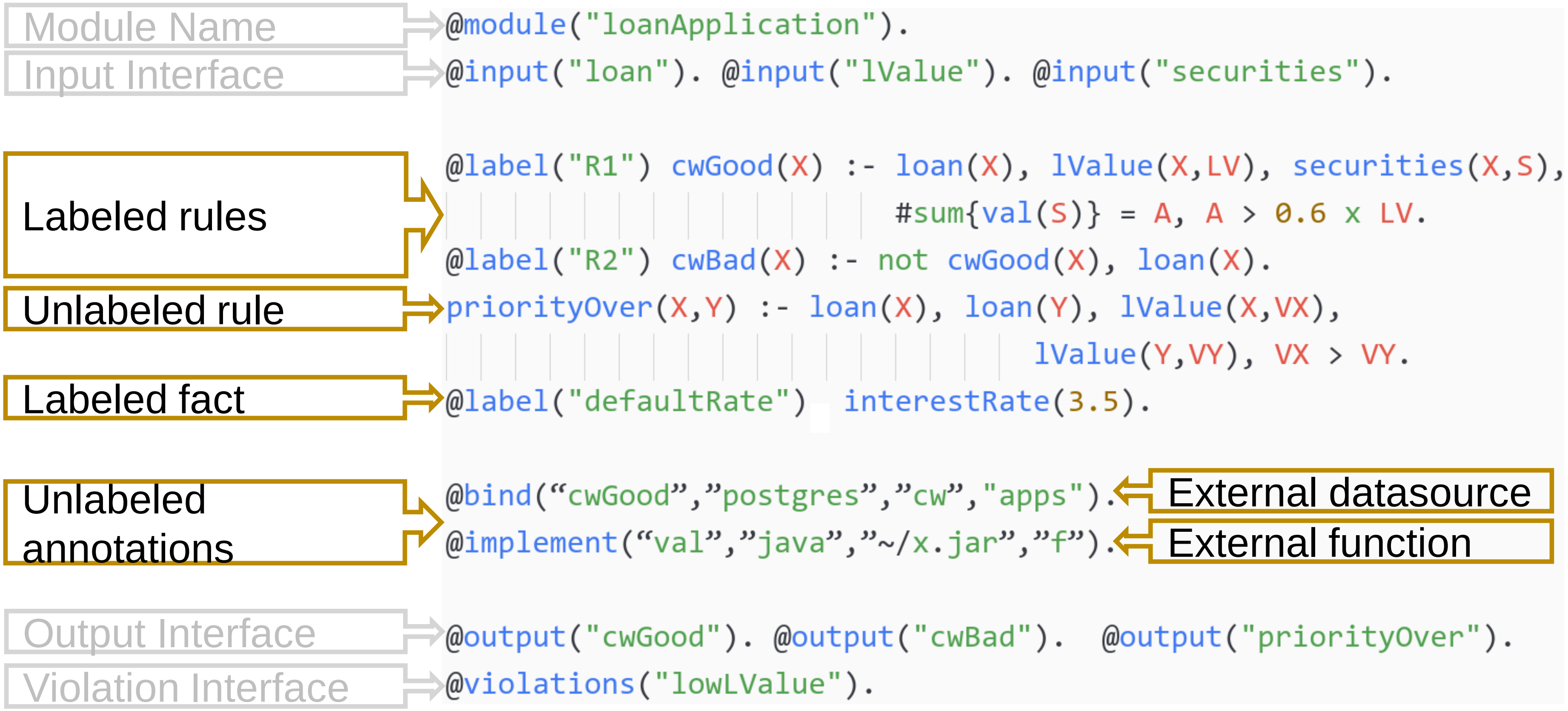
```
@module("loanApplication").
@input("loan"). @input("lValue"). @input("securities").

@label("R1") cwGood(X) :- loan(X), lValue(X,LV), securities(X,S),
                          #sum{val(S)} = A, A > 0.6 x LV.
@label("R2") cwBad(X) :- not cwGood(X), loan(X).
priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX),
                                  lValue(Y,VY), VX > VY.
@label("defaultRate")  interestRate(3.5).

@bind(“cwGood”,”postgres”,”cw”,"apps").
@implement(“val”,”java”,”~/x.jar”,”f”).

@output("cwGood"). @output("cwBad").  @output("priorityOver").
@violations("lowLValue").
```

BUSINESS RULE MODULES



Module Name

Input Interface

Labeled rules

Unlabeled rule

Labeled fact

Unlabeled annotations

External datasource

External function

Output Interface

Violation Interface

```
@module("loanApplication").
@input("loan"). @input("lValue"). @input("securities").

@label("R1") cwGood(X) :- loan(X), lValue(X,LV), securities(X,S),
                          #sum{val(S)} = A, A > 0.6 x LV.
@label("R2") cwBad(X) :- not cwGood(X), loan(X).
priorityOver(X,Y) :- loan(X), loan(Y), lValue(X,VX),
                                  lValue(Y,VY), VX > VY.
@label("defaultRate")  interestRate(3.5).

@bind(“cwGood”,”postgres”,”cw”,"apps").
@implement(“val”,”java”,”~/x.jar”,”f”).

@output("cwGood"). @output("cwBad").  @output("priorityOver").
@violations("lowLValue").
```

MOTIVATION

- Separate rule-based knowledge and applications
  → rule modules with interfaces
- Redundancy – rules applying in several settings
  → inheritance of rule modules
- Uncontrolled modification of inherited rules
  → modification restrictions

- Potential Application Areas
  - □ Business rule systems
  - □ Knowledge graph management
  - □ Rule-based information tailoring
  - □ Web data extraction
  - □ Internet of things