Implementing Datalog in Maude

M. A. Feliú
Universidad Politécnica de Valencia, DSIC / ELP

Joint work with María Alpuente, Christophe Joubert and A. Villanueva

San Sebastián, September 2009
Motivation

• Recent interest in defining complex interprocedural program analyses in few lines of Datalog
• Need of efficiently solving Datalog programs under a huge set of facts
• Existing approaches:
  ◦ **bddbbdd**: based on Bdds
  ◦ **Datalog_solve**: based on Bess
• Our proposal: from Datalog to maude
  ◦ remaining at a higher level
  ◦ aiming at handling metaprogramming features for the analysis
  ◦ keeping the need of being efficient
Outline

1. Datalog
2. Our proposal: from Datalog to Maude
3. Towards the transformation
   - The rule-based approach
   - The equational-based approach
4. Conclusions
Datalog

Relational language (similar to Prolog) using declarative clauses to both describe and query a deductive database

• Evaluation strategy:
  ◦ top-down (goal-directed) [Ullman 1985]
  ◦ bottom-up (inferred from base facts) [Ullman 1989]

• Datalog query: \( q = \langle G, R \rangle \) where:
  ◦ \( R \), a Datalog program defined over \( P, V \) and \( C \),
  ◦ \( G \), a goal.

• Additional restrictions
  ◦ Finite domains \( P, V \) (capital letters) and \( C \) (lower-case letters)
  ◦ Stratified Datalog programs
Datalog Example

Facts

supervise (Mary, Alice).
supervise (Alice, Mark).
Datalog Example

**Facts**

```
supervise (Mary, Alice).
supervise (Alice, Mark).
```

**Clauses**

```
superior (X, Y) :- supervise (X, Y).
superior (X, Y) :- supervise (X, Z), superior (Z, Y).
```
Datalog Example

Facts

supervise (Mary, Alice).
supervise (Alice, Mark).

Clauses

superior (X, Y) :- supervise (X, Y).
superior (X, Y) :- supervise (X, Z), superior (Z, Y).

Example of goal

:- superior (Mary, Y).
Datalog Example

Facts

supervise (Mary, Alice).
supervise (Alice, Mark).

Clauses

superior (X, Y) :- supervise (X, Y).
superior (X, Y) :- supervise (X, Z), superior (Z, Y).

Example of goal

:- superior (Mary, Y).

Answer

{Y/Alice, Y/Mark}
From Logic to Rewriting

- Transformations from logic to functional programming widely studied
  - Logic programming based on *resolution*
  - Functional programming based on *term rewriting*

- Idea: reusing the infrastructure of term rewriting to run the logic program preserving the observational behavior:
  - termination, success set, computer answers, …

- Traditionally, transformations impose an input/output relation among the parameters of the logic programs
  - restricting the way predicates may be invoked

- Giesl *et al.* provide a transformation not imposing a specific mode
  - preserving the termination behavior, not the computed answers
Our proposal

- The execution of the Maude specification recovers the computed answers of the original Datalog program
- Not moded transformation
The running example: Simple Pointer Analysis

Java-like program

o1 : p = new Object()

o2 : p = new Object()
The running example: Simple Pointer Analysis

Java-like program

```
o1 : p = new Object()
    .
    if ...
    .
    q = p
o2 : p = new Object()
    .
```

Datalog representation

```
vP0(p,o1).

a(q,p).
vP0(p,o2).
```
The running example: Simple Pointer Analysis

### Facts

- \( vP_0(p, o_1). \)
- \( vP_0(p, o_2). \)
- \( a(q, p). \)

### Clauses

- \( vP(V_1, H_1) :- vP_0(V_1, H_1). \)
- \( vP(V_1, H_1) :- a(V_1, V_2), vP(V_2, H_1). \)
The running example: Simple Pointer Analysis

**Facts**

- \( \text{vP0}(p, o1) \).
- \( \text{vP0}(p, o2) \).
- \( \text{a}(q, p) \).

**Clauses**

- \( \text{vP}(V1, H1) :\!\:- \ \text{vP}_0(V1, H1) \).
- \( \text{vP}(V1, H1) :\!\:- \ \text{a}(V1, V2), \ \text{vP}(V2, H1) \).

**Example of goal**

\[ :- \ \text{vP}(q, H). \]

**Answer**

\( \{H/o1, H/o2\} \)
The running example: Simple Pointer Analysis

Facts

\- vP0(p, o1).
\- vP0(p, o2).
\- a(q, p).

Clauses

\- vP(V1, H1) :- vP_0(V1, H1).
\- vP(V1, H1) :- a(V1, V2), vP(V2, H1).

Example of goal

\- vP(q, H).
\- vP(V, o1).

Answer

\{H/o1, H/o2\}
\{V/p, V/q\}
The running example: Simple Pointer Analysis

**Facts**

- vP0(p, o1).
- vP0(p, o2).
- a(q, p).

**Clauses**

- vP(V1, H1) :- vP0(V1, H1).
- vP(V1, H1) :- a(V1, V2), vP(V2, H1).

**Example of goal**

- :- vP(q, H).
- :- vP(V, o1).
- :- vP(q, o1).

**Answer**

- {H/o1, H/o2}
- {V/p, V/q}
- {} \equiv success

Implementing Datalog in Maude
The running example: Simple Pointer Analysis

**Facts**
- vP0(p, o1).
- vP0(p, o2).
- a(q, p).

**Clauses**
- vP(V1, H1) :- vP_0(V1, H1).
- vP(V1, H1) :- a(V1, V2), vP(V2, H1).

**Example of goal**
- :- vP(q, H).
- :- vP(V, o1).
- :- vP(q, o1).

**Answer**
- {H/o1, H/o2}
- {V/p, V/q}
- {} \(\equiv\) success
## The rule-based approach

<table>
<thead>
<tr>
<th>Datalog</th>
<th>Maude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Variables</td>
<td>Ground representation</td>
</tr>
<tr>
<td>Computed answers</td>
<td>Constraints</td>
</tr>
<tr>
<td>Non-determinism</td>
<td>Rules + search command</td>
</tr>
<tr>
<td>Unification</td>
<td>Pattern Matching + Consistency checking</td>
</tr>
</tbody>
</table>
Ground representation

sorts Variable Constant Term .
subsort Variable Constant < Term .

subsort Qid < Constant .
op v : Qid -> Variable [ctor] .
op v : Term Term -> Variable [ctor] .
Ground representation

sorts Variable Constant Term .
subsort Variable Constant < Term .

subsort Qid < Constant .
op v : Qid -> Variable [ctor] .
op v : Term Term -> Variable [ctor] .

• We construct variables \( v(T1,T2) \) to model existentially quantified variables
Datalog computed (partial) answer

- Answers are stored within the term representing the ongoing partial computation
- *Constraint*: set of equations representing the substitution of (logical) variables by (logical) constants

```
op _-_ : Term Constant -> Constraint .
```
Consistent computations

- In logic programming, unification takes care of the consistency of computations

```
op isConsistent : Constraint -> Bool .
eq isConsistent(success) = true .
ceq isConsistent((X = Cte1) , (X = Cte2) , C) = false
    if Cte1 =/= Cte2 .
ceq isConsistent((Cte1 = Cte2) , C) = false
    if Cte1 =/= Cte2 .
ceq isConsistent((T = Cte) , C) = true
    if isConsistent(C) [owise] .
```
Non-determinism

<table>
<thead>
<tr>
<th>DATALOG</th>
<th>MAUDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clauses</td>
<td>Conditional rules</td>
</tr>
<tr>
<td>Facts</td>
<td>Rules</td>
</tr>
<tr>
<td>Backtracking</td>
<td>search command</td>
</tr>
</tbody>
</table>
Clauses transformation

**Datalog clauses**

\[ vP(Var,Heap) :- vP0(Var,Heap). \]

**Maude code**

\[
\begin{align*}
\text{op vP vP0 a : Term Term -> Constraint .} \\
\text{crl vP(T1,T2) => C if} \\
\quad vP0(T1,T2) => C \land \\
\quad \text{isConsistent(C) .}
\end{align*}
\]
**Clauses transformation**

Datalog clauses

\[ vP(\text{Var1},\text{Heap}) :- a(\text{Var1},\text{Var2}), vP(\text{Var2},\text{Heap}). \]

Maude code

\[
\text{crl } vP(\text{T1},\text{T2}) \implies (v(\text{T1},\text{T2}) = \text{Cte}), C1, C2 \text{ if } \\
\text{ } a(\text{T1},v(\text{T1},\text{T2})) \implies ((v(\text{T1},\text{T2}) = \text{Cte}), C1) \land \\
\text{ } \text{isConsistent((v(\text{T1},\text{T2}) = \text{Cte}), C1)} \land \\
\text{ } vP(\text{Cte},\text{T2}) \implies C2 \land \\
\text{ } \text{isConsistent((v(\text{T1},\text{T2}) = \text{Cte}), C1, C2)}. 
\]
Facts transformation

<table>
<thead>
<tr>
<th>Datalog facts</th>
<th>Maude code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(q,p). vP0(p,o1). vP0(p,o2).</td>
<td>rl a(T1,T2) =&gt; ( T1 = 'q ), T2 = 'p .</td>
</tr>
<tr>
<td></td>
<td>rl vP0(T1,T2) =&gt; ( T1 = 'p ), T2 = 'o1 .</td>
</tr>
<tr>
<td></td>
<td>rl vP0(T1,T2) =&gt; ( T1 = 'p ), T2 = 'o2 .</td>
</tr>
</tbody>
</table>

For better efficiency...

Implementing Datalog in Maude
Facts transformation

Datalog facts
a(q,p). vP0(p,o1). vP0(p,o2).

Maude code
rl a(T1,T2) => ( T1 = 'q ), T2 = 'p .
rl vP0(T1,T2) => ( T1 = 'p ), T2 = 'o1 .
rl vP0(T1,T2) => ( T1 = 'p ), T2 = 'o2 .

For better efficiency...

crl a(T1,T2) => ( T1 = 'q ), T2 = 'p
   if (T1 == 'q or T1 :: Variable) and
   (T2 == 'p or T2 :: Variable).
Experimental results and conclusion

<table>
<thead>
<tr>
<th>Facts per predicate</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1.4 sec.</td>
<td>4.5 sec.</td>
<td>10.9 sec.</td>
</tr>
</tbody>
</table>

- Rules + backtracking + conditions penalize performance
The equational-based approach

**Rule-based**

- Rule non-determinism
- search
- Conditions guide execution
- Explicit consistency check

**Equational-based**

- Equation determinism
- reduce
- Unravelings guide execution
- *Implicit* consistency check

The computed term encodes the whole set of (partial) computed answers
Datalog computed (partial) answers

sorts Constraint EmptyConstraint NonEmptyConstraint TConstraint FConstraint.
subsort EmptyConstraint NonEmptyConstraint < Constraint.
subsort TConstraint FConstraint < EmptyConstraint.

op =_ : Term Constant -> NonEmptyConstraint.
op T : -> TConstraint.
op F : -> FConstraint.
op _,- : Constraint Constraint -> Constraint [assoc comm id: T].

• F represents a failed computation
Datalog computed (partial) answers

\begin{align*}
\text{var } &\text{ NEC } : \text{ NonEmptyConstraint} . \\
\text{var } &\text{ V } : \text{ Variable} . \\
\text{var } &\text{ Cte, Cte1, Cte2 } : \text{ Constant} . \\
\text{eq } &\text{ (Cte = Cte) = T . } \quad \text{--- Simplification} \\
\text{eq } &\text{ (Cte1 = Cte2) = F [owise] . } \quad \text{--- Unsatisfiability} \\
\text{eq } &\text{ NEC,NEC = NEC . } \quad \text{--- Idempotence} \\
\text{eq } &\text{ F,NEC = F . } \quad \text{--- Zero element} \\
\text{eq } &\text{ F,F = F . } \quad \text{--- Simplification} \\
\text{eq } &\text{ (V = Cte1),(V = Cte2) = F [owise] . } \quad \text{--- Unsatisfiability} \\
\end{align*}

- \textit{Implicit} consistency check
  - "\_\_\_" and "\_\_=\_" become defined symbols
Set of Datalog computed (partial) answers

sorts ConstraintSet EmptyConstraintSet NonEmptyConstraintSet .
subsort EmptyConstraintSet NonEmptyConstraintSet < ConstraintSet .
subsort NonEmptyConstraint NonEmptyConstraintSet < NonEmptyConstraintSet .
subsort FConstraint < EmptyConstraintSet .

op ;_ : ConstraintSet ConstraintSet ->
    ConstraintSet [assoc comm id: F] .

op ;_ : NonEmptyConstraintSet ConstraintSet ->
    NonEmptyConstraintSet [assoc comm id: F] .

var NECS : NonEmptyConstraintSet .

eq NECS ; NECS = NECS . --- Idempotence

- Non-deterministic datalog computations
  - "_;_" stands for disjunction
## Facts transformation

### Datalog facts

<table>
<thead>
<tr>
<th>Datalog facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(q,p).</td>
</tr>
<tr>
<td>vP0(p,o1).</td>
</tr>
<tr>
<td>vP0(p,o2).</td>
</tr>
</tbody>
</table>

### Maude code

<table>
<thead>
<tr>
<th>Maude code</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq a(T1,T2) = ((T1 = 'q), (T2 = 'p))</td>
</tr>
<tr>
<td>eq vP0(T1,T2) = ((T1 = 'p), (T2 = 'o1));</td>
</tr>
<tr>
<td>((T1 = 'p), (T2 = 'o2))</td>
</tr>
</tbody>
</table>

---

### Something new...

<table>
<thead>
<tr>
<th>op vP vP0 a : Term Term -&gt; ConstraintSet</th>
</tr>
</thead>
</table>
Datalog clauses

C1: \( vP(Var,Heap) :- vP0(Var,Heap). \)
C2: \( vP(Var1,Heap) :- a(Var1,Var2), vP(Var2,Heap). \)

Maude code

\[ eq \ vP(T1,T2) = vPc1(T1,T2) ; vPc2(T1,T2) . \]

- The solutions of a predicate are the disjunction of the solutions of its clauses
Clauses transformation

Datalog clauses

C1: vP(Var,Heap) :- vP0(Var,Heap).

Maude code

eq vPc1(T1,T2) = vP0(T1,T2) .
## Clauses transformation

### Datalog clauses

C2: \( vP(\text{Var1}, \text{Heap}) \) :- \( a(\text{Var1}, \text{Var2}), vP(\text{Var2}, \text{Heap}) \).

### Maude code

\[
\begin{align*}
\text{eq} & \quad vPc2(T1,T2) = vPc2s1(T1,T2) . \\
\text{eq} & \quad vPc2s1(T1,T2) = vPc2s2(a(T1,v(T1,T2)),T1,T2) . \\
\text{eq} & \quad vPc2s2(((v(T1,T2) = Cte) , C) ; CS,T1,T2) = \\
& \quad (vP(Cte,T1,T2) x ((v(T1,T2) = Cte) , C)) ; \\
& \quad vPc2s2(CS,T1,T2) . \\
\text{eq} & \quad vPc2s2(F,T1,T2) = F .
\end{align*}
\]

- “vPc2s1” and “vPc2s2” are unraveling functions which guide execution
Experimental results

One more thing...

op vP : Term Term -> ConstraintSet [memo].
Experimental results

One more thing...

\texttt{op vP : Term Term -> ConstraintSet [memo].}

<table>
<thead>
<tr>
<th>Facts/pred.</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>0.01 sec.</td>
<td>0.01 sec.</td>
<td>0.02 sec.</td>
<td>0.08 sec.</td>
<td>0.6 sec.</td>
</tr>
</tbody>
</table>
Conclusions and Future work

• Datalog for the analysis of OO programs
• Exploration of the influence of implementation choices on this kind of computation
• General transformation formalized
  ◦ equational-based
  ◦ not moded
  ◦ preserves computed answers
  ◦ correct and complete
• Future work
  ◦ Representing metaprogramming features for the analysis
  ◦ Improving efficiency (Datalog optimizations, . . . )
Questions?

For more information:
mfeliu@dsic.upv.es