

Datalog-based Program Analysis with BES and RWL



Christophe Joubert
Universidad Politécnica de
Valencia, DSIC / ELP

Joint work with María Alpuente,
Marco A. Feliú, Adam Keba,
Fernando Tarín and Alicia Villanueva

DATALOG 2.0, Oxford, UK
March 19, 2010

Motivation

- Concise and intuitive specifications of complex interprocedural program analyses in Datalog [Whaley-et-al-PLDI-04]
- Scalable BDD-based solver [Whaley-et-al-PLDI-04]
 - Problem: Simplicity is lost when handling JAVA reflection
- Datalog resolution algorithms with time and space guarantees [Liu-Stoller-TOPLAS-09]

The **ELP team** at **UPV** with expertise in:

- Boolean equation systems (BES)
 - Successful verification problems
 - Distributed BES resolution algorithms
- Rewriting logic (RWL) systems
 - Successful analysis problems
 - Efficient implementation in the high-level programming language Maude (metalevel)



Outline

- 1 Context: Pointer Analysis
- 2 From Datalog to BES [Alpuente-et-al-FMICS-08]
 - Transformation
 - DATALOG_SOLVE prototype
 - Summary
- 3 From Datalog to RWL [Alpuente-et-al-LOPSTR-09]
 - Transformation
 - DATALAUDE prototype
 - Summary
- 4 Conclusion



Outline

- 1 Context: Pointer Analysis
- 2 From Datalog to BES [Alpuente-et-al-FMICS-08]
 - Transformation
 - DATALOG_SOLVE prototype
 - Summary
- 3 From Datalog to RWL [Alpuente-et-al-LOPSTR-09]
 - Transformation
 - DATALAUDE prototype
 - Summary
- 4 Conclusion



Context: Pointer Analysis

Points-to analysis

- Disambiguate memory references in a program
- Answer to the question: "Which (abstract) memory locations might this reference-valued variable refer to at runtime?"
- Andersen's analysis (1994): Flow- and context-insensitive, inclusion-based points-to analysis for C programs

Memory elements in a program

- Memory references (`vP_0`)
- Memory writes (`store`)
- Memory reads (`load`)
- Assignments (`assign`)



Context: Pointer Analysis

Points-to analysis

- Disambiguate memory references in a program
- Answer to the question: "Which (abstract) memory locations might this reference-valued variable refer to at runtime?"
- Andersen's analysis (1994): Flow- and context-insensitive, inclusion-based points-to analysis for C programs

Memory elements in a program

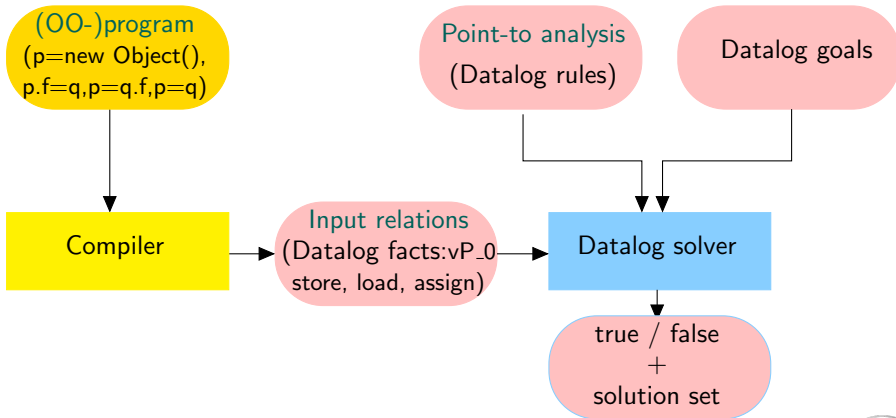
- Memory references (`vP_0`)
- Memory writes (`store`)
- Memory reads (`load`)
- Assignments (`assign`)

Memory elements deduced from analysis

- Memory references (`vP`)
- Memory locations (`hP`)



Datalog Analysis of Programs



Context-Insensitive (CI) Points-To Analysis

Only four Datalog rules

```
vP(V1, H1) :- vP_0(V1, H1).
```

```
vP(V1, H1) :- assign(V1, V2), vP(V2, H1).
```

```
hP(H1, F1, H2) :- store(V1, F1, V2), vP(V1, H1), vP(V2, H2).
```

```
vP(V2, H2) :- load(V1, F1, V2), vP(V1, H1), hP(H1, F1, H2).
```

Three finite domains

V (memory references), H (memory locations) and F (variable fields)

Two output relations

vP (variable : V, heap : H)

hP (base : H, field : F, target : H)



Example of CI Points-To Analysis of JAVA Program

JAVA program

```
public A foo {  
    ...  
    p = new Object(); /* o1 */  
    q = new Object(); /* o2 */  
    p.f = q;  
    r = p.f;  
    ...  
}
```



Example of CI Points-To Analysis of JAVA Program

JAVA program

```
public A foo {  
    ...  
    p = new Object(); /* o1 */  
    q = new Object(); /* o2 */  
    p.f = q;  
    r = p.f;  
    ...  
}
```

Datalog facts

```
vP_0(p, o1).  
vP_0(q, o2).  
store(p, f, q).  
load(p, f, r).
```



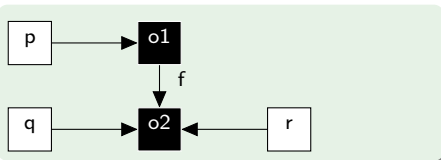
Example of CI Points-To Analysis of JAVA Program

JAVA program

```
public A foo {  
    ...  
    p = new Object(); /* o1 */  
    q = new Object(); /* o2 */  
    p.f = q;  
    r = p.f;  
    ...  
}
```

Datalog facts

```
vP_0(p, o1).  
vP_0(q, o2).  
store(p, f, q).  
load(p, f, r).
```



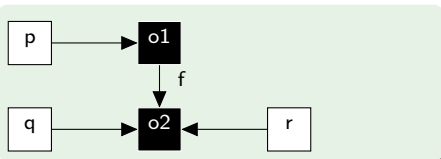
Example of CI Points-To Analysis of JAVA Program

JAVA program

```
public A foo {
    ...
    p = new Object(); /* o1 */
    q = new Object(); /* o2 */
    p.f = q;
    r = p.f;
    ...
}
```

Datalog facts

```
vP_0(p, o1).
vP_0(q, o2).
store(p, f, q).
load(p, f, r).
```



Example of goal

```
:- vP(r, Y).
```



Outline

- 1 Context: Pointer Analysis
- 2 From Datalog to BES [Alpuente-et-al-FMICS-08]
 - Transformation
 - DATALOG_SOLVE prototype
 - Summary
- 3 From Datalog to RWL [Alpuente-et-al-LOPSTR-09]
 - Transformation
 - DATALAUDE prototype
 - Summary
- 4 Conclusion



Related work

BES technology:

- Parameterised Boolean Equation Systems (PBES) more expressive and generic than Datalog queries
 - First-order value-based μ -calculus model checking
 - Equivalence checking of infinite LTS modulo bisimulation
 - Known linear time and memory BES resolution methods

Well-adapted to demand-driven analyses:

- Best during program development
 - Errors discovered with only a partial exploration of the program
- Reduction of Horn Clauses satisfiability to boolean graph resolution [Liu-Smolka-ICALP-98]
- Concise representation of imperative graph/sets algorithms (e.g., [Liu-Stoller-TOPLAS-09])



Parameterised BES (alternation-free)

Many verification problems ...

- Model-checking of μ -calculus
- Equivalence checking
- Partial-order reduction
- Test case generation
- Data-flow analyses:
 - Live/dead variables
 - Very busy expressions
 - Available expressions
 - Reachable definitions
 - Influence variables, ...
- **Datalog query evaluation**



Parameterised BES (alternation-free)

Many verification problems ...

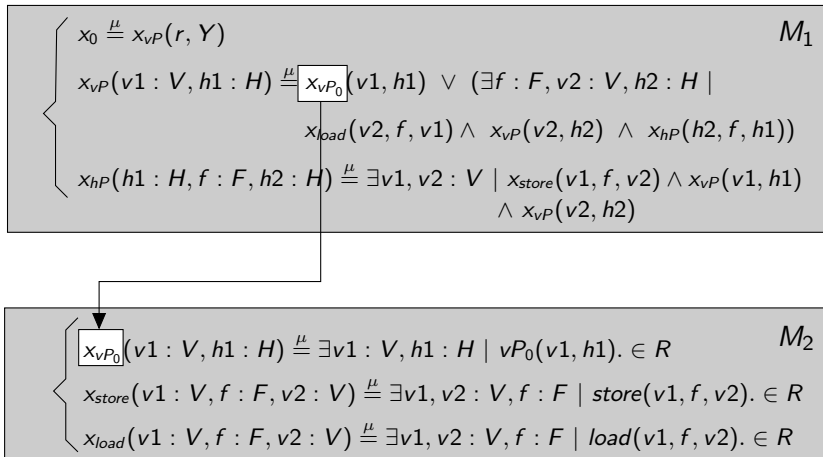
- Model-checking of μ -calculus
- Equivalence checking
- Partial-order reduction
- Test case generation
- Data-flow analyses:
 - Live/dead variables
 - Very busy expressions
 - Available expressions
 - Reachable definitions
 - Influence variables, ...
- Datalog query evaluation

... one generic solution:

- Translation to a boolean equation system (BES) resolution
- Use of generic BES solver and diagnostic generation



Parameterised Boolean Equation Systems



(Top-down) Transformation from Datalog to (P)BES

Given a Datalog query $q = \langle G, R \rangle$, there exists a PBES, whose boolean variables x_i are in one-to-one correspondence with predicates of q , and are defined as follows:

$$x_0 \stackrel{\mu}{=} \bigvee_{p_1(d_1), \dots, p_m(d_m). \in G} \bigwedge_{i=1}^m x_{p_i}(d_i) \quad (1)$$

$$\{x_p(d : D) \stackrel{\mu}{=} \bigvee_{p(d) :- p_1(d_1), \dots, p_m(d_m). \in R} \bigwedge_{i=1}^m x_{p_i}(d_i) \mid p \in P\} \quad (2)$$



(P)BES Evaluation

Principle

Datalog query evaluation reduced to the resolution of variable x_0

Resolution of x_0

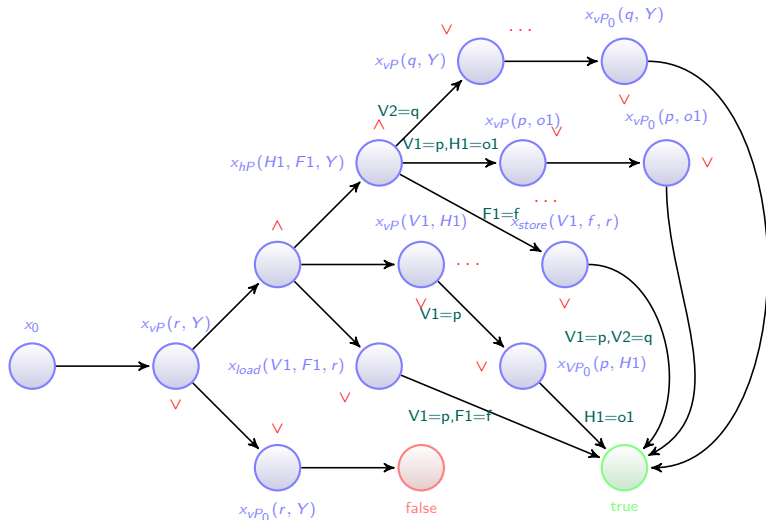
- **Incremental** expansion of the P_{BES} starting from x_0
- **Finite** number of generated new variable instances from x_0
- Expanded P_{BES} portion converted into a **plain BES**
- Value of x_0 obtained by **locally** solving the resulting BES

Remark

Incremental expansion and local resolution done **simultaneously**



Translation of CI Analysis Rules into BES Resolution



(Bottom-up) Transformation from Datalog to (P)BES

$$\begin{aligned}
 X_0 & \stackrel{\nu}{=} \bigwedge_{\forall Q(Cs) \in \text{givenFacts}} X_{1,Q(Cs)} \\
 X_{1,P(Zs,As)} & \stackrel{\nu}{=} \bigwedge_{\forall P(Zs,As) \rightarrow Q1(Z's,Bs)} X_{1,Q1(Z's,Bs)} \\
 \wedge \bigwedge_{\forall P(X1s,Ys,As) \wedge P2(X2s,Ys,C2s) \rightarrow Q2(X1s,X2s,Y's,C3s)} X_{1,Q2,\{X1s,X2s,Y's\},C3s} \\
 & \quad Zs = \{X1s,Ys\}, P Ys X1s \{Ys\} \cup = X1s, \forall X2s | P2 Ys X2s \{Ys\} \\
 \wedge \bigwedge_{\forall P1(X1s,Ys,C1s) \wedge P(X2s,Ys,As) \rightarrow Q2(X1s,X2s,Y's,C3s)} X_{1,Q2,\{X1s,X2s,Y's\},C3s} \\
 & \quad Zs = \{X2s,Ys\}, P Ys X2s \{Ys\} \cup = X2s, \forall X1s | P1 Ys X1s \{Ys\}
 \end{aligned}$$

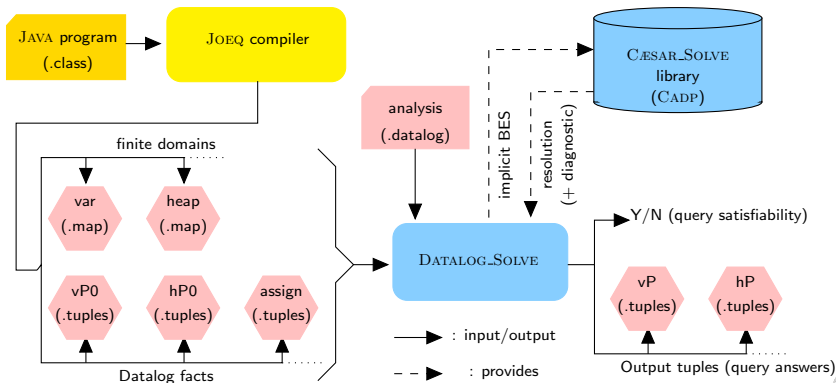
For the simple pointer analysis (2 rules), the size of the specialized resolution algorithm is:

- 4 boolean equations (instantiation of the BES formalism above)
- vs. 35 lines of imperative algorithm in [Liu-Stoller-TOPLAS-09]

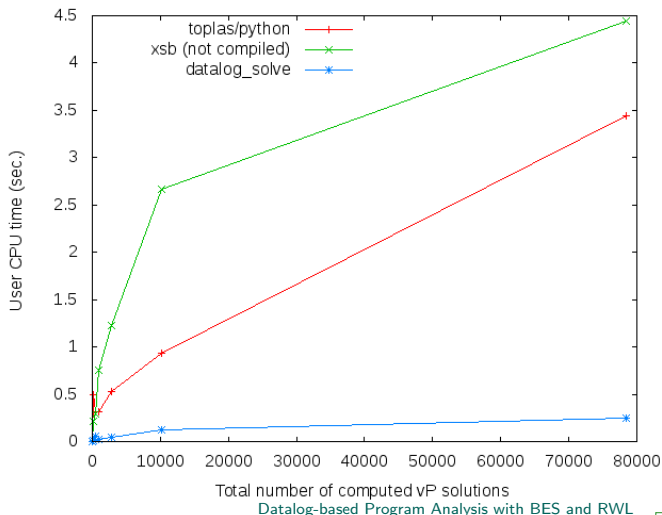


DATALOG_SOLVE Prototype Architecture

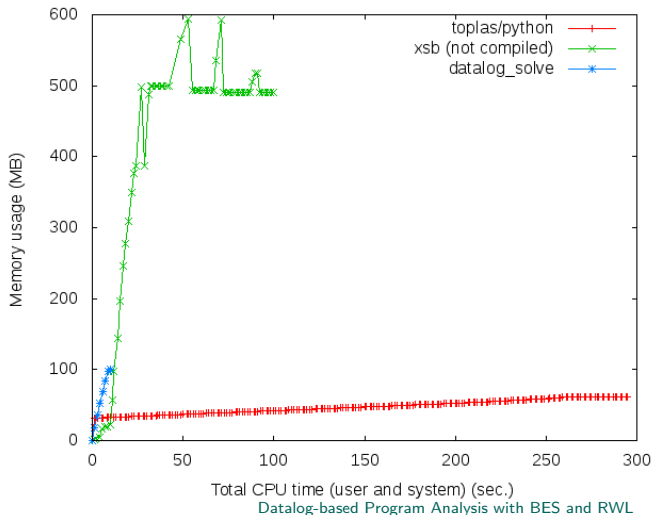
- v1 and v2 (top-down): 120 lines of LEX, 380 lines of BISON and 3500 lines of C code
- v3 (bottom-up): 716 lines of C code (2 days of work)



Analysis time on random inputs



Memory usage on a random input with $6 \cdot 10^6$ solutions

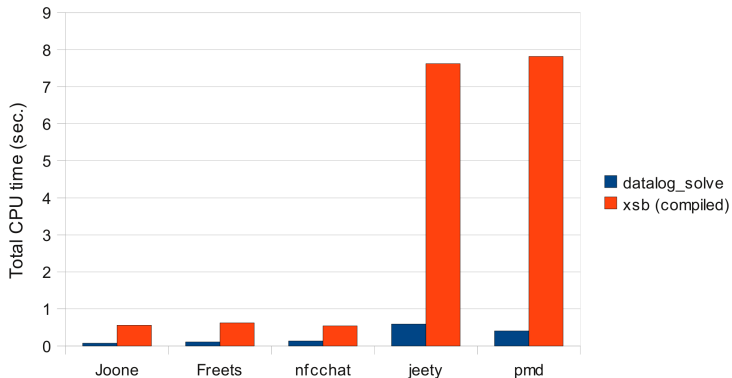


Real JAVA program inputs

Name	Description	Classes	Methods	Vars	Locations
freetts (1.2.1)	speech synthesis	215	723	8K	3K
nfcchat (1.1.0)	chat client	283	993	11K	3K
jetty (6.1.10)	servlet container	309	1160	12K	3K
joone (2.0.0)	neural net	375	1531	17K	4K
pmd (4.2.2)	code analyzer	394	1971	19K	4K



Analysis time of the CI points-to analysis



From Datalog to BES: Summary

- New application of the BES technology to logic programs
- Straightforward transformation into demand-driven BES resolution
- Time and memory resolution linear in the BES size
- `DATALOG_SOLVE`, new component of the `CADP` toolset
- Application to demand-driven and bottom-up interprocedural pointer analysis of real-size `JAVA` programs
- Future work
 - Distribution of analysis over interconnected workstations (`CADP`)



Outline

- 1 Context: Pointer Analysis
- 2 From Datalog to BES [Alpuente-et-al-FMICS-08]
 - Transformation
 - DATALOG_SOLVE prototype
 - Summary
- 3 From Datalog to RWL [Alpuente-et-al-LOPSTR-09]
 - Transformation
 - DATALAUDE prototype
 - Summary
- 4 Conclusion



Motivation

- Remaining at a higher level (unified declarative framework)
 - Some approaches are not purely declarative (use of JAVA) [Livshits-et-al-APLAS-05].
- Existing transformations from Logic Programming to Rewriting [Marchiori-ALP-94, Reddy-SLP-84, Schneider-Kamp-et-al-LOPSTR-06]
- Efficient RWL implementation with interesting features in Maude
 - higher expressivity
 - ACU (associative-commutative-unity)
 - efficient implementation
 - reflection/metaprogramming
- Final goals:
 - expressing analysis involving reflection concisely
 - being efficient enough



From Logic to Rewriting

- Different operational mechanisms:
 - Logic programming based on *resolution*
 - Functional programming based on *term rewriting*
- Transformations preserve the observational behavior:
 - termination, success set, computed answers, . . .
- We don't want to impose a specific mode
 - Traditional transformations impose an input/output relation among the parameters
 - [Schneider-Kamp-et-al-LOPSTR-06](#) provides a non-moded transformation
 - preserves the termination behavior, not the computed answers



Intuition of the Transformation

What we want...

Query $\xrightarrow{*}$ Answer Constraint Set

- We want a query to progressively reduce to an answer constraint set

What we must consider...

1. Ground representation
2. Non-determinism
3. Unification
4. Guided execution



Comparison of RWL Approaches

RULE-BASED

Rules (non-determinism)

`search`

Conditions guide execution

Explicit consistency check

Backtracking

→

→

→

→

→

EQUATIONAL-BASED

Equations (determinism)

`reduce`

Unravelings guide execution

Implicit consistency check

No backtracking (+memoization)

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



Rule-based Transformation of a Simple Analysis

Datalog clauses

```
vP(Var,Heap) :- vP0(Var,Heap).
vP(Var1,Heap) :- a(Var1,Var2), vP(Var2,Heap).
```

Maude code

```
cr1 vP(T1,T2) => C if
    vP0(T1,T2) => C /\ isConsistent(C) .

cr1 vP(T1,T2) => (v(T1,T2) = Cst) , C1 , C2 if
    a(T1,v(T1,T2)) => ((v(T1,T2) = Cst) , C1) /\
        isConsistent((v(T1,T2) = Cst) , C1) /\
    vP(Cst,T2) => C2 /\
        isConsistent((v(T1,T2) = Cst) , C1 , C2) .
```



Equational-based Transformation of a Simple Analysis

Datalog clauses

$vP(\text{Var}, \text{Heap}) :- vP0(\text{Var}, \text{Heap}). \quad (C1)$

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

Maude code

eq $vP(T1, T2) = vPc1(T1, T2) ; vPc2(T1, T2) .$

eq $vPc1(T1, T2) = \mathbf{vP0}(T1, T2) .$

eq $vPc2(T1, T2) = \mathbf{vPc2s1}(T1, T2) .$

eq $\mathbf{vPc2s1}(T1, T2) = \mathbf{vPc2s2}(a(T1, v(T1, T2)), T1 T2) .$

eq $vPc2s2(((v(T1, T2) = Cst) , C) ; CS, T1 T2) =$

eq $\mathbf{vPc2s2}(((v(T1, T2) = Cst) , C) ; CS, T1 T2) =$

$(vP(Cst, T2) \times ((v(T1, T2) = Cst) , \neg C)) ;$

$\mathbf{vPc2s2}(CS, T1 T2) .$

eq $\mathbf{vPc2s2}(F, T1 T2) = F .$



Complex Analyses: Reflective Pointer Analysis

Reflection in JAVA

Technique to examine or modify the runtime behavior of applications running in the JAVA virtual machine. Based on `java.lang.reflect`.

Example of use

Write to object fields and invoke methods that are not known at compile time.

Generally handled in an unsound or ad-hoc manner in static analysis approaches



Simple example of JAVA reflection

```
class P0 {  
  P0 (String c1, String c2) {  
    this.c1 = c1;  
    this.c2 = c2;  
  }  
  public String c1;  
  public String c2;  
}
```

```
public class Main {  
  public static void main(String[] args) {  
    P0 u = new P0("", "");  
    String v = "c1";  
    String w = "c2";  
    java.lang.reflect.Field r = P0.class.getField(v);  
    r.set(u, w);  
    v = u.c1;  
  }  
}
```

A flow-insensitive points-to reflective analysis would give the following result: **r** points to **u.c1** and **u.c2**.

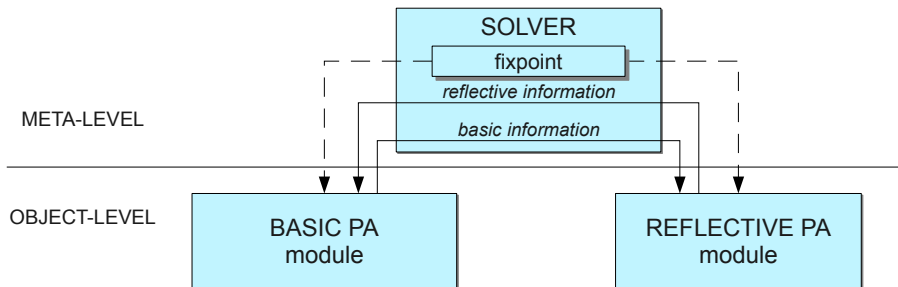


Analysis input extracted from the JAVA program

Java Code	Extracted Information	Reflective Information
<code>P0 u = new P0("", "");</code>	<code>vP0(u,0).</code> <code>vT(u,P0).</code>	
<code>String v = "c1";</code>	<code>vP0(v,12).</code> <code>vT(v,string).</code>	<code>stringToField(12,c1).</code>
<code>String w = "c2";</code>	<code>vP0(w,15).</code> <code>vT(w,string).</code>	<code>stringToField(15,c2).</code>
<code>java.lang.reflect.Field r = P0.class.getField(v);</code>	<code>vP0(\$0,18).</code> <code>vT(\$0,ClassP0).</code> <code>vT(r,field).</code> <code>mI(main,21,getField).</code> <code>iRet(21,r).</code> <code>actual(21,0,\$0).</code> <code>actual(21,1,v).</code>	<code>getField(Class.getField).</code>
<code>r.set(u, w);</code>	<code>mI(main,30,set).</code> <code>actual(30,0,r).</code> <code>actual(30,1,u).</code> <code>actual(30,2,w).</code>	
<code>v = u.c1;</code>	<code>l(u,c1,v).</code>	



Structure of the reflective analysis



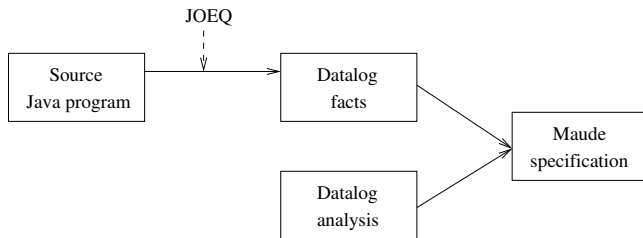
- Basic program analysis module (11 rules)
- Reflective program analysis module (11 rules)
- Solver module (50 rules)

⇒ 100% declarative (RWL) specification and analysis



DATALAUDE Prototype Architecture

- Transformation script: 950 lines of Haskell
- Example of generated output: 2 rules/7 equations in Maude
(pa.maude, 2-rules points-to analysis)



Running time for the simple pointer analysis example

Memoization (for free in Maude)

```
op vP : Term Term -> ConstraintSet [memo].
```



Running time for the simple pointer analysis example

Memoization (for free in Maude)

```
op vP : Term Term -> ConstraintSet [memo].
```

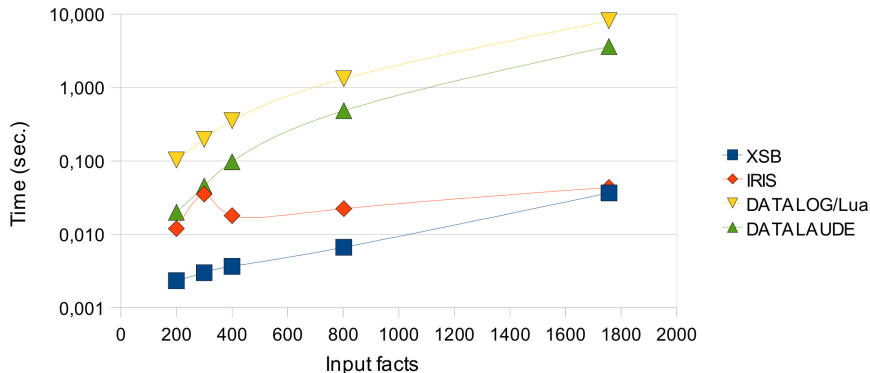
Comparison between three rewriting approaches implemented in DATA LAUDE:

a/2	vP0/2	vP/2	rule-based	equational	eq.+memo
100	100	144	6.0	0.67	0.02
150	150	222	20.59	2.23	0.04
200	200	297	48.48	6.11	0.10
403	399	602	382.16	77.33	0.47
807	1669	2042	4715.77	1098.64	3.52



Running time for the pointer analysis example

Comparison between four Datalog implementations



From Datalog to RWL: Summary

- Datalog for the analysis of OO programs
- Two general transformations have been formalized and implemented
 - rule- and equational-based
 - not moded
 - preserves computed answers
 - correct and complete
- Representing reflective features for the analysis (Maude capability)
- Specification by modules: composition of analysis on demand
- **Future work**
 - Using a more compact representation of facts and rules
 - Considering other optimizations (memoization at the logical level, TOPLAS-09, etc.)



Outline

- 1 Context: Pointer Analysis
- 2 From Datalog to BES [Alpuente-et-al-FMICS-08]
 - Transformation
 - DATALOG_SOLVE prototype
 - Summary
- 3 From Datalog to RWL [Alpuente-et-al-LOPSTR-09]
 - Transformation
 - DATALAUDE prototype
 - Summary
- 4 Conclusion



Conclusion and Future Work

- Summary



- BES technology well adapted to:
 - Fixed-point computation
 - Demand-driven analyses
 - Parallelization
- RWL well adapted to:
 - Declarative, accurate and sound analysis specification and resolution
 - Metatheory of program analysis

- Ongoing and future work



- Time and space guaranteed distributed Datalog resolution
- Unified RWL framework (extended with compact rule representation)



Bibliography

-  M. Alpuente, M.A. Feliú, C. Joubert and A. Villanueva.
Using Datalog and Boolean Equation Systems for Program Analysis.
FMICS'2008, LNCS 5596:215–231.
-  M. Alpuente, M.A. Feliú, C. Joubert and A. Villanueva.
Defining Datalog in Rewriting Logic.
LOPSTR'2009, LNCS 6037.

Tool descriptions:

-  M. Alpuente, M.A. Feliú, C. Joubert and A. Villanueva.
DATALOG_SOLVE: A Datalog-Based Demand-Driven Program Analyzer.
ENTCS 248:57–66.
-  M. Alpuente, M.A. Feliú, C. Joubert and A. Villanueva.
Implementing Datalog in Maude.
PROLE'2009, 15–22.



Questions?



For more information:

joubert@dsic.upv.es

www.dsic.upv.es/~joubert

DATALOG_SOLVE available online:

http://safe-tools.dsic.upv.es/datalog_solve

DATALAUDE available online:

<http://safe-tools.dsic.upv.es/datalaude>