Static Analysis of JAVA Programs in a Rule–based Framework

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

Joint work with María Alpuente, Christophe Joubert and Alicia Villanueva

PROLE 2008
Gijón
October 8, 2008
Outline

1. What's the tool?
2. What's the basis?
3. What's under the hood?
4. What are the results?
5. Conclusion
**What’s the tool?**

**Datalog Solve**: a Datalog solver

- **Datalog** is well suited for concisely expressing complex program analyses
- There exists efficient **Bes** resolution algorithms
- **Datalog Solve** transforms **Datalog** queries into **Bes**
- In particular, **Datalog Solve** can be used to solve **Java** pointer analyses
What’s the basis?

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?"

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**

Relational language (similar to Prolog) using declarative rules 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 set of *goals*

- **Additional restrictions**
  - Stratified DATALOG programs
Datalog analysis of programs

(Java) program
(p=new Object(), p.f=q, p=q.f, p=q)

Compiler

Input relations
(Datalog facts: vP_0 store, load, assign)

Datalog solver

true / false + computed answers

Points-to analysis

Analysis specification (Datalog rules)

Analysis invocation (Datalog goals)

Static Analysis of JAVA Programs in a Rule–based Framework
Example of **Datalog** Points-to analysis

**Input Relations**

- \(vP_0(p, o1)\)
- \(vP_0(q, o2)\)
- \(store(p, f, q)\)
- \(load(p, f, r)\)

**Points-to Analysis Specification**

- \(vP(V1, H1) :- vP_0(V1, H1)\).
- \(hP(H1, F1, H2) :- store(V1, F1, V2), vP(V1, H1), vP(V2, H2)\).
- \(vP(V2, H2) :- load(V1, F1, V2), vP(V1, H1), hP(H1, F1, H2)\).

**Example of Datalog goal**

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

**Answer**

\(vP(r, o2) \leftrightarrow \{Y/o2\}\)
**Datalog_Solve Architecture**

- **Datalog_Solve**: 120 lines of **Lex**, 380 lines of **Bison** and 3,500 lines of **C** code

Static Analysis of JAVA Programs in a Rule–based Framework
What’s under the hood?

**DATALOG-SOLVE**

- **Datalog Query**
  
  \[ q = \langle G, R \rangle \]

- **Datalog to Pbes transformer**

- **Pbes**

- **Pbes to Bes transformer**

- **Implicit Bes resolution**
What’s under the hood?

CÆSAR_SOLVE library

- On-the-fly resolution of alternation-free Bess.
- Developed in CADP using OPEN_CÆSAR.
- 4 linear-time sequential algorithms (10,000 lines of C)
  - DFS and BFS for general Bess
  - DFS memory-efficient for acyclic or conjunctive/disjunctive Bess
- 1 linear-time distributed algorithm (10,000 lines of C)
- Diagnostics (boolean subgraphs)
- Generic, application-independent
What’s under the hood?

**CADP**

- One of the leading verification toolboxes in academia
- Offers various (> 42) tools for
  - visualization
  - simulation
  - equivalence checking
  - testing
  - model checking
- Open platform supporting integration of other specification, verification and analysis techniques (> 29 academic tools integrated)
- Originally designed for verifying correctness of LOTOS specifications
**CADP and Datalog_Solve**

What’s the tool? What’s the basis? What’s under the hood? What are the results? Conclusion

[Diagram showing the relationship between tools and concepts]

- **LotOS**
- **LTS**
- **Traces**

- **CAESAR.OPEN**
- **BCG.OPEN**
- **SEQ.OPEN**

- **Datalog**

- **OPEN_CÆSAR API** (implicit LTS)

- **CÆSAR_SOLVE** (implicit Bes)

- LTS generation, Interactive Simulation, On-the-fly verification, Test generation, Datalog query evaluation

Static Analysis of JAVA Programs in a Rule–based Framework
**Datalog** Specification of a Points-To Analysis

### Domains

- $V$ 262144 variable.map
- $H$ 65536 heap.map
- $F$ 16384 field.map

### Relations

- $vP_0$ (variable : $V$, heap : $H$) inputtuples
- $store$ (base : $V$, field : $F$, source : $V$) inputtuples
- $load$ (base : $V$, field : $F$, dest : $V$) inputtuples
- $assign$ (dest : $V$, source : $V$) inputtuples
- $vP$ (variable : $V$, heap : $H$) outputtuples
- $hP$ (base : $H$, field : $F$, target : $H$) outputtuples

### Rules

- $vP (v, h) :- vP_0 (v, h)$.
- $vP (v1, h) :- assign(v1, v2), vP (v2, h)$.
- $hP (h1, f, h2) :- store(v1, f, v2), vP (v1, h1), vP (v2, h2)$.
- $vP (v2, h2) :- load (v1, f, v2), vP (v1, h1), hP (h1, f, h2)$.

---

Static Analysis of JAVA Programs in a Rule–based Framework
Experiments

- Context-insensitive points-to analysis
- Four Java projects (~ 300 classes) from sourceforge.net:
  - freetts (1.2.1): speech synthesis system (freetts.sourceforge.net)
  - nfcchat (1.1.0): scalable, distributed chat client (nfcchat.sourceforge.net)
  - jetty (6.1.10): server and servlet container (jetty.sourceforge.net)
  - joone (2.0.0): Java neural net framework (joone.sourceforge.net)
- Metrics
  - Running time of analysis
  - Peak memory usage of analysis
- Runtime environment: JAVA JRE 1.5, JOEQ version 20030812, Intel Core 2 T5500 1.66GHz, 3GB RAM, Linux Kubuntu 8.04
Experimental Results

• Description of the JAVA benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Classes</th>
<th>Methods</th>
<th>Vars</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>freetts (1.2.1)</td>
<td>215</td>
<td>723</td>
<td>8K</td>
<td>3K</td>
</tr>
<tr>
<td>nfcchat (1.1.0)</td>
<td>283</td>
<td>993</td>
<td>11K</td>
<td>3K</td>
</tr>
<tr>
<td>jetty (6.1.10)</td>
<td>309</td>
<td>1160</td>
<td>12K</td>
<td>3K</td>
</tr>
<tr>
<td>joone (2.0.0)</td>
<td>375</td>
<td>1531</td>
<td>17K</td>
<td>4K</td>
</tr>
</tbody>
</table>

• Time and peak memory usage

<table>
<thead>
<tr>
<th>Name</th>
<th>time (sec.)</th>
<th>memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>freetts (1.2.1)</td>
<td>10</td>
<td>61</td>
</tr>
<tr>
<td>nfcchat (1.1.0)</td>
<td>8</td>
<td>59</td>
</tr>
<tr>
<td>jetty (6.1.10)</td>
<td>73</td>
<td>70</td>
</tr>
<tr>
<td>joone (2.0.0)</td>
<td>4</td>
<td>58</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

• Summary
  ○ New application of the BES technology to logic programs
  ○ Transformation into demand-driven BES resolution
  ○ Time and memory resolution linear in the BES size
  ○ DATALOG\_SOLVE, new component of the Cadp toolset for evaluating DATALOG queries, namely demand-driven interprocedural pointer analysis of real-size Java programs
  ○ Easily applicable to other languages

• Ongoing and future work
  ○ Better BES resolution algorithms with recent DATALOG optimizations (time and space guarantees)
  ○ Distribution of analysis over interconnected workstations
Details on the formalization

María Alpuente, Marco A. Feliú, Christophe Joubert and Alicia Villanueva.
Using Datalog and Boolean Equation Systems for Program Analysis.

Questions?

For more information:
mfeliu@dsic.upv.es

DATALOG_SOLVE available online:
www.dsic.upv.es/users/elp/datalog_solve