Interactive, Scalable, Declarative Program Analysis: From Prototype to Implementation

William C. Benton
University of Wisconsin

willb@cs.wisc.edu
http://pages.cs.wisc.edu/~willb/

Joint work with Charles N. Fischer
Logic programming is an expressive and effective tool for program analysis.
Logic programming is an expressive and effective tool for program analysis.
Logic programming is an expressive and effective tool for program analysis.
Logic programming is an expressive and effective tool for program analysis.
Four activities in analysis design
Four activities in analysis design
Four activities in analysis design
Four activities in analysis design
Four activities in analysis design
Four activities in analysis design
null
Our contribution: DIMPLE

- Analysis framework for Java bytecodes
- Flexible enough for interactive use
- Fast enough for production
- Techniques scale to realistic programs
DIMPLE supports each activity!
Tabled Prolog

- Some predicates use Prolog-style search
- Some predicates are *tabled*
  - Stores subgoal results in a trie
  - Identifies SCCs, finds finite fixed points
- Bounded term-size implies termination
Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen’s analysis
Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen’s analysis
soot → DimpleGenerator

DIMPLE code generator → Yap Prolog → DIMPLE internal rules
Java bytecodes → soot → DimpleGenerator

DIMPLE code generator → Yap Prolog → DIMPLE internal rules
Java bytecodes → soot → DimpleGenerator

- Analysis rules
- Statement processing rules

DIMPLE code generator → Yap/Prolog → DIMPLE internal rules

DIMPLE IR
Java bytecodes → soot → DimpleGenerator

Analysis rules

Statement processing rules

DIMPLE IR

DIMPLE code generator

Yap Prolog

DIMPLE internal rules
Java bytecodes

soot

DimpleGenerator

Analysis rules

Statement processing rules

DIMPLE code generator

Yap Prolog

DIMPLE internal rules

DIMPLE IR

Queries

Results

William C. Benton
The DIMPLE IR

- Describes whole program & libraries
  - Class hierarchy
  - Control flow
  - Conservative call graph
  - Every statement
DIMPLE IR statement example

```prolog
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```
DIMPLE IR statement example

Program counter

stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')), 
        intConstant(1)))).
DIMPLE IR statement example

Program counter

stmt(units_736, assignStmt(
local(locals_730, method(SIG), primtype('boolean')), 
intConstant(1))).
DIMPLE IR statement example

Program counter

```
stmt(units_736, assignStmt(local(locals_730, method(SIG), primtype('boolean')), intConstant(1))).
```

Statement structure

Local `local(locals_730, method(SIG), primtype('boolean')), intConstant(1))`. 
DIMPLE IR statement example

Program counter

```plaintext
stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1))).
```

Statement structure

Local

```plaintext
local(locals_730, method(SIG), primtype('boolean'))
```
DIMPLE IR statement example

Program counter

stmt(units_736,
    assignStmt(
        local(locals_730, method(SIG), primtype('boolean')),
        intConstant(1)))).

Statement structure

Local

Method signature

e.g. '<java.lang.Object: equals(Ljava/lang/Object;)Z>'
DIMPLE IR statement example

Program counter

stmt(units_736,
assignStmt(
   local(locals_730, method(SIG), primtype('boolean')), 
   intConstant(1))).

Statement structure

Local

Method signature
e.g. '<java.lang.Object: equals(Ljava/lang/Object;)Z>'

Type
Forecast

• Overview of the DIMPLE system
• Designing preprocessors and analyses
• Interactive development
• Case study: Andersen’s analysis
Statement processing rules

Head <-- Body
Statement processing rules

allocSite(PC, Type) <--
    stmt(PC, assignStmt(local(L,M,T), newExpr(Loc,Type))).

reference(L) <--
    local(L,M,T), T \= primtype(P).

explicitlyThrows(Sig,T) <--
    containsStmt(Sig, PC),
    stmt(PC, throwStmt(local(L,M,T))).
Statement processing rules

\[
\text{allocSite}(\text{PC}, \text{Type}) \leftarrow \text{stmt}(\text{PC}, \text{assignStmt}(\text{local}(L,M,T), \text{newExpr}(\text{Loc}, \text{Type}))).
\]

\[
\text{reference}(L) \leftarrow \text{local}(L,M,T), T \neq \text{primtype}(P).
\]

\[
\text{explicitlyThrows}(\text{Sig}, T) \leftarrow \text{containsStmt}(\text{Sig}, \text{PC}), \text{stmt}(\text{PC}, \text{throwStmt}(\text{local}(L,M,T))).
\]
Statement processing rules

\[
\text{allocSite}(PC, \text{Type}) \leftarrow \text{stmt}(PC, \text{assignStmt}(\text{local}(L,M,T), \text{newExpr}(\text{Loc}, \text{Type}))).
\]

\[
\text{reference}(L) \leftarrow \text{local}(L,M,T), T \neq \text{primtype}(P).
\]

\[
\text{explicitlyThrows}(\text{Sig}, T) \leftarrow \text{containsStmt}(\text{Sig}, PC), \text{stmt}(PC, \text{throwStmt}(\text{local}(L,M,T)))).
\]
Statement processing rules

\[
\text{allocSite}(PC, \text{Type}) \leftarrow \text{stmt}(PC, \text{assignStmt}(\text{local}(L,M,T), \text{newExpr}(\text{Loc}, \text{Type}))).
\]

\[
\text{reference}(L) \leftarrow \text{local}(L,M,T), T \not\in \text{primtype}(P).
\]

\[
\text{explicitlyThrows}(\text{Sig}, T) \leftarrow \text{containsStmt}(\text{Sig}, PC), \text{stmt}(PC, \text{throwStmt}(\text{local}(L,M,T)))).
\]
Statement processing rules

- Act like filters on program database
- Body may contain arbitrary Prolog code
- Generate extra debugging information
Analysis rules

\[ \text{Head} \iff \text{Body} \]
Analysis rules
Analysis rules

\[
x_r(a,b) \iff r(a,b).
\]

\[
x_r(a,b) \iff x_r(a,i), r(i,b).
\]
Analysis rules

• Act like predicates over relations
• Typically execute as tabled code
  • User-directed option to disable
• Support tracing and debugging
Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen’s analysis
Interactive development

- Whole program is available for queries
- From intuition to hypothesis to analysis
- Test and refine ideas, define new rules
Query example

“Most field accesses are to/from this.”

stmt(Id, assignStmt(local(A,B,C),
    instanceFieldRef(local(L,M,T), F))),
isthis(L).

stmt(Id, assignStmt(local(A,B,C),
    instanceFieldRef(local(L,M,T), F))),
\+ isthis(L).
Forecast

- Overview of the DIMPLE system
- Designing preprocessors and analyses
- Interactive development
- Case study: Andersen’s analysis
Andersen’s analysis

• Points-to analysis
• Flow- and context-insensitive
• Subset-based
Andersen’s analysis

\[pc: r = \text{new } C();\]
\begin{align*}
\text{r} & \quad \text{o}_{pc} \\
\text{r} & \quad \text{x} \\
\text{o}_i & \quad f & \quad \text{x} & \quad \forall i : \text{pt}(r, o_i) \\
\text{r} & \quad f & \quad \text{o}_i & \quad \forall i : \text{pt}(x, o_i)
\end{align*}
\[ v_{pt}(Ref, Id) \leq v_{pt}(Ref, Id) \]

\[ v_{pt}(Ref, Id) \leq \text{assign}(Ref, Refl), v_{pt}(Refl, Id) \]

\[ v_{pt}(Ref, Id) \leq \text{s\_load}(Ref, F), \text{s\_store}(F, Refl), v_{pt}(Refl, Id) \]

\[ v_{pt}(Ref, Id) \leq \text{formal}(Ref, I, M), \text{actual}(Refl, I, M), v_{pt}(Refl, Id) \]

\[ h_{pt}(Obj1, F, Obj2) \leq\]
\[ \text{store}(Ref1, F, Ref2), v_{pt}(Ref1, Obj1), v_{pt}(Ref2, Obj2) \]

\[ v_{pt}(Ref, Id) \leq\]
\[ \text{load}(Ref1, F, Ref), v_{pt}(Ref1, Id1), h_{pt}(Id1, F, Id) \]

\[ v_{pt}(Ref, Id) \leq\]
\[ \text{ret\_caller}(Ref, Method), \text{ret\_callee}(Refl, Method), v_{pt}(Refl, Id) \]

(Rules adapted from Whaley and Lam, 2004.)
Andersen’s analysis: performance results

<table>
<thead>
<tr>
<th>input</th>
<th>relations</th>
<th>outputs (× $10^6$)</th>
<th>CPU time (s.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>347,290</td>
<td>12.31</td>
<td>8.13</td>
</tr>
<tr>
<td>bloat</td>
<td>476,635</td>
<td>19.41</td>
<td>13.54</td>
</tr>
<tr>
<td>chart</td>
<td>586,337</td>
<td>4.74</td>
<td>3.88</td>
</tr>
<tr>
<td>eclipse</td>
<td>401,474</td>
<td>12.06</td>
<td>11.91</td>
</tr>
<tr>
<td>hsqldb</td>
<td>304,451</td>
<td>6.64</td>
<td>4.56</td>
</tr>
<tr>
<td>luindex</td>
<td>322,634</td>
<td>7.32</td>
<td>7.42</td>
</tr>
<tr>
<td>pmd</td>
<td>427,279</td>
<td>10.67</td>
<td>7.01</td>
</tr>
<tr>
<td>xalan</td>
<td>302,150</td>
<td>6.18</td>
<td>4.13</td>
</tr>
</tbody>
</table>
Summary

- Tabled Prolog is useful for analysis tasks
- DIMPLE: a framework for analysis
  - Logic programming used throughout
- Suitable from prototype to production
Any questions?

http://pages.cs.wisc.edu/~willb/dimple/