Fast and Precise Points-to Analysis

Jonas Lundberg, Tobias Gutzmann, and Welf Löwe

Växjö University, Sweden

September 29, 2008

Motivation

- Points-to analysis: (Static) dataflow analysis
  - Which objects can variable v possibly reference during program execution?
  - Compute the *points-to set* $Pt(v) =$ set of abstract objects v may reference
  - Abstraction: Map possible *runtime objects* $\longrightarrow$ *abstract objects*
    - usually: group objects created at the same *syntactic location* together
- Provides input data for, e.g., escape analysis, virtual call resolution
- Goals: high precision, fast execution

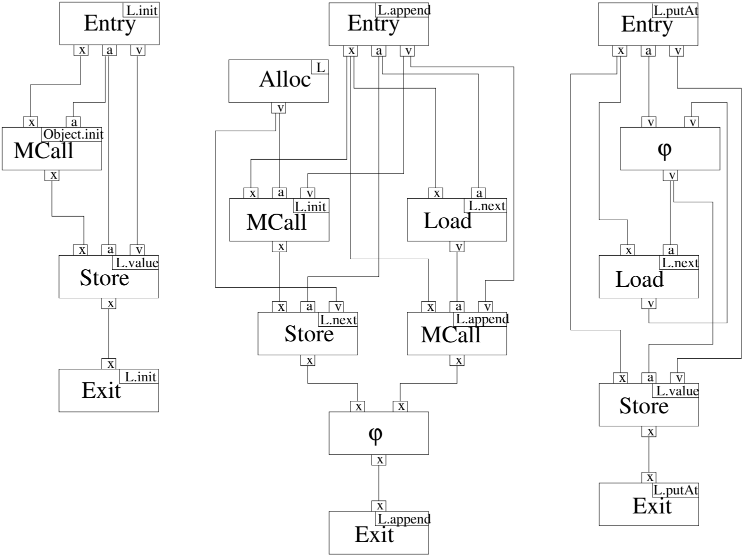
Our approach

- Static Single Assignment (SSA) form based
- Simulated execution: inter- och intra-procedural flow-sensitivity
- *this*-sensitivity: our new context-senstive approach, which is much faster and almost as precise as the well-known *object*-sensitivity

Points-to SSA

- Our graph-based SSA program representation, designed especially for Points-to analysis
- non-pointer related operations are removed, e.g., operations related to primitive types
- variables are resolved to edges in the graph
- all dependencies are explicit
- → allows ordering of operations → local flow sensitivity

Points-to SSA - example



Simulated Execution

- Simulation of the actual execution of a program
- Start at one or more entry methods
  - interrupt the analysis when a call expression occurs
  - follow the call → continue analyzing the potentially called methods
  - resume with the calling method once analysis of the called method(s) is completed
- → inter- och intra-procedural flow-sensitivity

Context Sensitivity

- Distinguish different invocations of a method depending on calling context
- Analyze method for each context separately
- Calling context:
  - call site - from where is the method called?
  - functional - depending on current analysis state

This-Sensitivity

- Our new functional approach to context-sensitivity.
- Contexts distinguished by the points-to set $Pt(a)$
- In comparison: The well known *object-sensitive* approach analyzes a target call for each $o \in Pt(a)$
- Too similar to be new?

This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$

This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$
- this-sensitivity:
    - need to analyze $foo()$ twice
    - both calls analyzed under different contexts

This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$
- this-sensitivity:
  - need to analyze *foo*() twice
  - both calls analyzed under different contexts
- object-sensitivity:
  - need to analyze *foo*() five times
  - in total three different contexts

This-Sensitivity vs. Object-Sensitivity

- two (unrelated) calls: $a_1.m()$ and $a_2.m()$
- let $Pt(a_1) = \{o_1, o_2\}$, $Pt(a_2) = \{o_1, o_2, o_3\}$
- this-sensitivity:
  - need to analyze *foo*() twice
  - both calls analyzed under different contexts
- object-sensitivity:
  - need to analyze *foo*() five times
  - in total three different contexts
- There are obviously big differences between the two approaches

This-Sensitivity vs. Object-Sensitivity

- Precision: We can show that neither approach is strictly more precise than the other (→ paper)
- Analysis cost:
    - this-sensitivity has, in theory, exponential analysis cost (as there may be 2^n contexts for each method, in regard to the number of abstract objects)
    - that doesn't seem to happen in practice! (And we could easily implement a fail-safe)
    - object-sensitivity: multiple targets for each call

Results - Metrics

- Three precision metrics we present here:
- *OEdge* and *Enter*:
  - a low number means better precision for *side effect analysis*, *escape analysis* etc.
- *PCall*:
  - a low number means better precision for *virtual call resolution*
- We have some more metrics in the paper

Results - Analysis Precision

	ThisSens			ObjSens		
Program	PCall	OEdge	Enter	PCall	OEdge	Enter
antlr	1.00	0.22	0.70	1.00	0.16	0.70
javadoc	0.99	0.42	0.69	1.00	0.41	0.68
emma	0.97	0.15	0.44	0.97	0.15	0.44
obfusc	0.99	0.61	0.63	0.99	0.50	0.63
...						
average	**0.97**	**0.44**	**0.59**	**0.97**	**0.32**	**0.59**
median	**0.99**	**0.42**	**0.63**	**1.00**	**0.24**	**0.63**

- Results indicate analysis precision relative to *context insensitive* analysis.
- this-sensitivity is comparably precise to object-sensitivity, except for the *OEdge* metric
- Other metrics (not on this slide) strengthen the observation that precision is comparable

Results - Analysis Cost

		ThisSens		ObjSens	
Program	Classes	Context	Time	Context	Time
antlr	225	3.36	0.97	3.91	4.65
javadoc	416	4.61	1.33	10.04	12.65
emma	749	3.91	0.63	11.63	9.15
obfusc	688	2.73	1.23	3.36	3.67
...					
average		**4.15**	**1.08**	**7.48**	**11.1**
median		**3.45**	**1.14**	**5.37**	**9.97**

- *Classes* is the number of classes in the program → input size (does not include library classes)
- *Context* is the avarage number of contexts per method → memory requirement metric
- *Time* is the analysis time as a factor to context insensitive analysis

Conclusion

- We have presented our flow-sensitive Points-to analysis
- New context sensitive approach to Points-to analysis: *this-sensitivity*
- Exponential analysis cost in theory
- Almost as fast as context insensitive analysis *in practice*
- Experiments show:
  - Almost as precise as object-sensitivity
  - But much, much faster in practice