Pointer Analysis in the Presence of Dynamic Class Loading

Martin Hirzel, Amer Diwan and Michael Hind

Presented by

Brian Russell

Claim: First nontrivial pointer analysis dealing with all Java language features including:

- Dynamic class loading
- Reflection
- Native methods

Optimizations that could benefit:

- Inlining
- Load elimination
- Code motion
- Stack allocation
- Parallelization

Dynamic class loading vs Static analysis

- Where is loaded class coming from?
- Which class will be loaded?
- When will the class be loaded?
- Will the class even be loaded at all?

An extension of Andersen’s analysis

- Type based analysis: fast but imprecise
- Shape analysis: more precise but slower
- Choosing between ~linear time type based analysis and Andersen’s slower analysis for connectivity-based garbage collection.
- Andersen’s had more challenges …

Some definitions

- Online interprocedural analysis – done at execution time.
- Demand-driven interprocedural analysis – static analysis limited to specified portions of code. Scalable.
- Incremental interprocedural analysis – avoids complete reanalysis after changes to code.
- Extant analysis – analysis done to code that is not affected by dynamic class loading.

Construction on an online algorithm

- Static approach is to scan program, build call graph.
- Online approach discovers parts of the program as parts of events during execution.
- Events include: startup, class loading, method compilation, reflection execution, native code execution and type resolution.
- Online additions to Andersen's algorithm include: online call graph construction, repropagation support, unresolved type support and capture of input events.

Constraint graph nodes

- h node -- heap object associated with an allocation site.
- v node -- a static variable or all instances of a local (stack) variable.
- h.f node – instance field f of all heap objects represented by h.
- v.f node – instance of a field f of H nodes pointed to by v.
- Significance: models instances, not declarations!

Intraprocedural constraints

- FlowTo sets – flow of values stored in v nodes and v.f nodes.
- FlowFrom sets – inverse of FlowTo sets.
- Points-to sets – set of R-values that a pointer may point to. Stored in v nodes and h.f nodes. Field sensitive nodes are more precise.
- Generated from assignment statements and heap allocations.

Interprocedural contraints

- Parameters, return values
- CHA finds call edges online when both caller and callee are compiled.
- Exceptions have values that flow from throw site to invoked catch clause. Could improve on assuming that all catch clauses are caught with type filtering or by limiting to callers.

Constraint propagator

- When v node points-to list is changed, add to worklist.
- When h.f node workflow set has changed and needs point-to list propagated.
- Goal is to limit points-to set propagation to sets that have actually changed.
- Experimented with collapsing single entry subgraphs and partial online cycle elimination.

Dealing with unresolved types

- Problem: CHA only deals with complete type info, but dynamic class loading invariably means some types are unresolved.
- Solution: Type resolution manager holds unresolved type info and notifies nodes when the type is resolved.

Dealing with reflection

- A new feature in Java 2.
- The Class class has methods that provide metainformation about other classes to Java programs.
- Choice of what gets invoked influenced by class metainformation.
- Compile time analysis impossible.
- Constraints generated at runtime.

Native code

- How does a JVM analyze code witten in other languages?
- By examining the JNI API to the code.
- Precision through type filtering.

Validation methodology

- Tied checks to garbage collection.
- GC graph traversal matches pointers in points-to sets? Yes -> good. No -> bad, print warning.
- Found some bugs.

Uses of online constraints

- Method inlining.
- Connectivity based garbage collection.

Performance

- Characteristics of programs in benchmark suite not explained (except null).
- Percentage of application execution time spent analyzing methods quite low, i.e. not many times normal execution.
- No info on real or potential speedup of benchmark programs.
- FYI: cubic time cost and quadratic time cost of online algorithm.