History

► 1.0 2002

Interprocedural Program Analysis

Prof. Uwe Aßmann
RISE
Contents

► Preliminaries
► Interprocedurally invalid paths
► Ways to solve an interprocedural problem
  ▶ Ignorant
  ▶ Functional
  ▶ k-String
    ▶ Interprocedural Control Flow Graph
  ▶ Lazy cloning
► The PAG tool
► Outlook
What is a Procedure?

- A named lambda abstraction
  - With parameters
  - A return instruction that returns a return value
- A schema for a runtime instance, an activation record
- An abbreviation for code that is called from different reference points (call sites)
- Abstractions with parameters can be found in any specification or programming language
  - Z, ..
  - Generic classes in Generic Java
- Procedures form the component model of the chip
Abstract Interpretation

► Symbolic execution of the program with symbolic or abstract values
  ▶ Since the values cannot be concrete we must abstract them to "easier" values, i.e., simpler domains, of finite count, height, or breadth
► Complete partial orders, semi-lattices or lattices are used as domains
► The lattice expresses the “unknown”, i.e., the unknown decisions at control flow decision points (if's)
  ▶ In the following called L
Functions:

- $\alpha : D \rightarrow L$, abstraction function from concrete to abstract
- $\gamma : L \rightarrow D$, concretization function from abstract to concrete
- $f^# : L \rightarrow L$, abstract semantic function (flow/transfer function)
Functions:

- $\alpha$ abstraction function, $\gamma$ concretization function, and $f^\#$ abstract semantic function must form an "almost" commuting diagram.

- They must be "interchangeable", formally: a Gaulois connection: $f \subseteq \gamma f^\# \alpha$.

  - The concrete semantics must be a subset of the concretization of the abstract semantics (conservative approximation).
  
  - The abstract semantic value is a superset of the concrete semantic value after application of the transfer function.
  
  - The concrete value of $f$ must be a subset of the abstracted value after application of the transfer function.
Sets of Transfer Function Equations

For an abstract interpretation, for all nodes in the control flow graph, set up transfer functions

\[ \{ f_n : L \rightarrow L \} \]

\[ \iff \]

\[ f_1 : L \rightarrow L \]

\[ \ldots \]

\[ f_k : L \rightarrow L \]
Intraprocedural Coincidence Theorem

- [Kam/Ullman]
- For all \( n: \text{Node} \): \( \text{MFP}(n,f_n) = \text{MOP}(n,f_n) \)
- The maximum fixpoint of an iterative evaluation of the system of transfer functions \( f_n \) at a node \( N \) is equal to the value of the meet over all paths to a node \( n \) (\( \text{MOP}(n) \))
- Means
  - No matter how the functions are iterated, if they stop, they stop at the meet over all paths
  - Any iteration algorithm can be used to reach the abstract values at each node (i.e., the maximal fixpoint of the function system)
Worklist Algorithm As Example

- Iterating *forward* over a worklist that contains “nodes not finished”

  - worklist = nodes;
  - While worklist ≠ NIL do
    - SELECT n FROM worklist;
    - // forward propagation
    - X := meet( f#(v) ) where v in n.predecessors[ControlFlowGraph];
    - IF X ≠ v(n) THEN
      - v(n) = X; worklist += n.successors[ControlFlowGraph];
    - END
Interprocedural Control Flow Graphs and Valid Paths

- Flow Functions $f#$
  - Nodes $f#(n)$
  - or Edges $f#(e)$

- "interprocedural edges": call edges from caller to callee

- Local edges from "call" to "return"

- Problem: not all interprocedural paths will be taken at the run time of the program
  - Call can return are symmetric
  - From whereever I enter a procedure, to there I leave
Interprocedural Problems

- Non-valid interprocedural paths invalidate the coincidence for the interprocedural case
- Knoop found a restricted one [CC92]:
  - No global parameters of functions
  - Restricted return behavior
Different Approaches to Interprocedural Analysis
Invalidating Approach

- All information is invalidated during a call
  - After the call, worst case value is assumed (top of lattice)
- Conservative (know nothing)
- Improvement:
  - Invalidate everything that might be written by the callee
  - However then alias analysis must run before
The Cloning/Inlining Approach

- Copy a procedure's body for every call and propagate information separately in body (builds up a interprocedural control flow graph, ICFG)
- Corresponds to inlining into every callee
- Leads to code (information) bloat
- Is space-exponential in nesting depth of call graph

```
quicksort
```

```
quicksort
  quicksort
```

```
quicksort
```
Expand the ICFG

quicksort

quicksort-1

quicksort-2

quicksort-1

quicksort

quicksort

quicksort-2

quicksort

quicksort
The Functional Approach

- Also called *effect calculation approach*
- Calculates a function/effect $F$ for every procedure $f$
  - Which is applied to the current input values at a caller to receive the output values after the call
  - Symbolic execution with an "abstract" function $F$
- $F$ is stored in a table, mapping abstract input value to abstract output value (i.e., an associative array of abstract values)
- Whenever the analysis reaches the callee, the current abstract input value is looked up
  - If found, reuse output value
  - Otherwise reanalyze body
The k-Call Context Approach

- Also called *k-call string approach*
- The call history of the called procedure is incorporated in the underlying lattice $D$ (*call strings*)
- Different bodies at different call sites are distinguished by the call strings
- In case of $k=1$ all call sites are distinguished
- $K=2$: all call sites, with calling context of callers
- $K=3$: all call sites, all calling contexts of the grandfathers
- ...
Expanded Supergraphs

- Abstract Values are copied for every caller (multiplicity)
- Connectors connect the right incarnation of the value to a caller site
- Example
  - mult(n) = 1 ==> no call sites are distinguished
  - mult(Pi) = ki where i is number of call sites ==> call string length 1
  - mult(Pi) = ki*n ==> call string length n
The Lazy Cloning Approach

- Agesen: Type inference for SELF
- Idea: do not clone
- During propagation, store all input values of functions analyzed so far
- If an input value for a function differs from an already memoized one, clone the parameter (i.e., distinguish it)
- Cloning parameters only
- Cloning them on demand
- Cloning can be restricted
  - Analysis works less precise but costs less memory
The Interprocedural Phi-Approach

- M. Trapp (Optimization of object oriented programs) introduces interprocedural phi functions (i-phi)
- i-phis are "small-ifs" or "ifs for one value"
- Every formal parameter of a procedure gets as input an i-phi
- The i-phi depends on the control flow condition
- More in some days from now...
PAG

- Intra and interprocedural analysis
- Extended super graph for interprocedural case
- Languages
  - Specification of the intermediate representation
  - Lattice
  - Abstract/Flow/transfer functions
Analyzer at Run Time
Node Orderings

- Nodes are ordered in the worklist. Different orderings are possible:
  - DFS: depth-first
  - BFS: breadth-first
  - SCC-D: strongly connected components in visit order depth first.
  - SCC-B: same in breadth first
  - WTO-D: SCCs, but ordered in weak topological ordering of Bourdoncle. Depth-first.
  - WTO-B: same, but breadth-first
Data Type Specifications

- **Basic sets**
  - Snum, unum, real, chr, string

- **Basic Lattices**
  - Lsnum, lunum, bool, a..b, enum

- **Type constructors**
  - Disjoint sum
  - Tuple construction *
  - Powerset operator
  - List operator
  - Function on S1 -> S2
Lattice Specifications

- flat(Set S)
- lift(Lattice L)
- powerset(Set S)
- Tuple space
- Function space (function lattice) S->L, pointwise ordering
- dual(Lattice L)
- reduce(Lattice E, reduction function f)
- 3 different implementations
- .. Examples ..
Example Live Variables

GLOBAL
    maxvar: snum

SET
    vars = [0..maxvar]

LATTICE
    varset = set(vars)
    var = lift(varset)
Example Caches

GLOBAL
storeMin: unum
storeMax: unum
cacheSize: unum
aWays: unum<24

SET
storeLine = [storeMin..storeMax]
direct = [0..cacheSize]

LATTICE

cacheLine = [0..aWays]
age = lift(cacheLine)
assoc = storeLine -> age
cache = direct -> assoc
dfi = cache * cache
Example Intervals

LATTICE

upperBound = Isnum
lowerBound = dual(Isnum)
interv = lowerBound * upperBound
env = snum -> interv  // variables to intervals
dom = lift(env)
Example Heap Analysis

LATTICE

\[
\begin{align*}
\text{node} &= \text{set}(\text{sn}) & & \text{// nodes abstract vars} \\
\text{edge} &= \text{node} \times \text{sn} \times \text{node} \\
\text{edges} &= \text{set}(\text{edge}) \\
\text{sedge} &= \text{sn} \times \text{node} \\
\text{sedges} &= \text{set}(\text{sedge}) \\
\text{shared} &= \text{set}(\text{node}) & & \text{// predicate} \\
\text{graph} &= \text{sedges} \times \text{edges} \times \text{shared} \\
\text{dfi} &= \text{lift}(\text{graph})
\end{align*}
\]
Types of the nodes of the CFG can be specified.

- Constructor based
- With alternatives

SYNTAX

START: Unlabstat

Unlabstat: M_Assign(var:Var, exp:Exp)
    | M_While(exp:Exp, body:Stat*)

...
Specification of Transfer Functions

- Similar to ML
- Pattern matching on IR nodes
- Functions are annotated to control flow graph nodes
  - Implicit parameter @ for data flow value
  - Return a value
- Dynamic Functions (updatable)
  - Application f({!x!})
  - Updating f[n->v]
  - Constant function [->v]
Specification of Transfer Functions

- Lattices
  - Glb, lub, <, >
- For latted and lifted lattices
  - drop, lift
- ZF Zermelo-Fränkel Set Expressions
  - \([x \sqcup x \leftarrow \text{set, if } x \geq 0]\)
Example:

// while(id <= exp)
M_While(M_Binop(M_op_leq(),
    M_Var_exp(M_simpl_var(id)),
    exp),_),true_edge):

let f <= @;
    id = val-Identifier(id);
in
let erg = f{!id!} glb (top,
    (eval(exp,f))!2);
in if is_ok(erg) then lift(f[!id->erg])=
    else bot;
endif;
Other Parts of the Specification

- Direction forward/backward
- Carrier
- Combine function (e.g., least-upper bound lub)
- Init value
- Init_start init value of start node
- Equal: equality test for fixpoint detection
- Widening function
- Narrowing function
Example

PROBLEM interval
  direction: forward
  carrier: dom
  init_start: lift([->(dual(0),0)])
  widening: wide
  narrowing: narrow
Debugging Specifications

► Export to VCG file format (or aiSee)
► Many visualizations possible
► Specific ones for flow graphs
  ► Lattice values annotated without edges to the nodes or edges of the flow graph
  ► Zoom in/out
  ► Hiding relations
  ► Blocks of nodes as regions with different color
AiSee (Successor to VCG)
AiSee (Successor to VCG)
AiSee (Successor to VCG)
AiSee (Successor to VCG)
Ressources

  - www.absint.de (also aiSee)
  - www.cs.uni-sb.de/~ martin/pag
The End