Constant Propagation

Discover values that are constant on all possible executions, and propagate values

Undecidable ➔ Conservative methods

Can create new constants in process by constant folding

Ex.

Entry
read(A)
B := 3
C := 4*B
C > B ➔ read(A)

D := B + 2
E := A + B
write(E)
Exit

Benefit?

Lattice for Constant Propagation

May be constant

true false...c0 c1 c2 ...

Constant

Not constant

any ⊕ ⊤ = any
any ⊕ ⊥ = ⊥

cli ⊕ cj = \{ cli if i=j

⊥ if i≠j

Initial value: ⊤
Simple Constant Propagation (Kildall)

Propagate constants through CFG, each branch equally likely

Worklist algorithm

$V_1 := V_2 + V_3$

$O(E \cdot N^2)$

Out($V_1$) = eval ($l_2 + l_3$)

Out($V_i$) = In($V_i$) for $i=1$

eval($l_2 + l_3$) =

\[ \begin{cases} 
  l_2 + l_3 & \text{if both are constants} \\
  \text{ow if } l_2 \text{ or } l_3 \text{ is } \text{ow} & \\
  \text{ow if } l_2 \text{ or } l_3 \text{ is } \text{ow} 
\end{cases} \]

Simple CP Example

Entry

A := 2

B := 3

A < B

C := 4

C := 5

Y

Exit
**Sparse Simple CP (Reif & Lewis)**

Propagate constants through SSA graph (def-use chains from SSA) for expected efficiency (size of SSA graph)

Worklist algorithm

Assign lattice values to expressions $e$

- $e$ not evaluated at compile time: \[ \text{eval}(e) \]
- no variables in $e$
- o.w.

Initialize WL to set of SSA edges where def is not

Algorithm terminates when WL is empty

---

**Sparse Simple CP (continued)**

Take SSA edge off WL

$V_1 := V_1, V_2, V_3, ..., V_n$

$l_1, l_2, l_3, ..., l_n$

$e := \text{Use}(V_1)$

$V_1, V_2, V_3, ..., V_n$

$j_1, j_2, j_3, ..., j_n$

Compute $n_1 = \text{if } j_1 \neq j_1$ set $j_1 = n_1$

Evaluate $e$; if its value has changed, add to WL all SSA edges from $e$’s node
Sparse Simple CP Example

Entry

A := 2

B := 3

A < B

C1 := 4

C2 := 5

C3 := phi(C1, C2)

Exit

Conditional Constant Prop. (Wegbreit)

Propagate constants through CFG, taking branches into account
Performs CP + Unreachable code elimination
More powerful than SCP or SSCP

Ex.

I := 1

I > 1

J := 1

J := 2

J must be 2

Executable Edges:

Determined by symbolic execution, starting from Entry

If node with single successor executed, that successor edge is marked executable
If predicate is executed, evaluate predicate and determine branch taken, if possible
Conditional CP (continued)

Worklist algorithm:

Each edge is initialized to be non-executable
The edge from Entry is marked executable, and put on WL

Proceed by symbolic execution:

\[ V1 := \]

\[ \text{Exec} \]

\[ \text{Becomes Exec} \]

\[ \text{Exec} \]

\[ \text{Eval(Pred)} \]

\[ \text{then all branches exec.} \]

\[ \text{true} \text{ then true branch exec.} \]

\[ \text{false} \text{ then false branch exec.} \]

\[ O(E \times N^2) \]

Conditional CP Example

Entry

\[ A := 2 \]

\[ B := 3 \]

\[ A < B \]

\[ C := 4 \]

\[ C := 5 \]

Exit
Sparse Conditional CP (Wegman, Zadeck)

Like Conditional CP but uses SSA graph for efficiency
Like Sparse Simple CP but meet applied only to arguments which correspond to executable incoming edges to Phi function

$O(\# \text{ Flow edges} + \# \text{ SSA edges})$

Assumes only 1 statement or predicate per node

Worklist Algorithm

- **FlowWL** flow graph edge work list, initialized to edges out of Entry
- **SSAWL** SSA edge work list, initialized to empty set
- **ExecFlag(e)** records whether flow graph edge $e$ is executable
- **LatCell(V,B)** lattice value for variable $V$ at node $B$

Sparse Conditional CP Algorithm

*Do while (FlowWL nonempty or SSA WL nonempty)*

*Take edge $e$ from WL.*

- If $e$ is a flow edge, then if it is executable, do nothing
  - else Set ExecFlag to true
    - Call VisitPhi for all Phi functions at $e$'s target
    - If this is the first time the target node is visited, Call VisitExp
      - if the target node has 1 out flow edge, add to WL
  - If $e$ is a SSA edge, and its target has a Phi function, call VisitPhi
    - if its target is an expression, and an incoming flow edge to it is executable, then Call VisitExp

*ow do nothing*
Sparse Conditional CP Algorithm, cont.

VisitPhi: Called when value of Lattice Cell of operand is lowered, or when edge becomes executable

\[ V_4 := \Phi(V_1, V_2, V_3) \]

Compute \( l_4 \) by taking meet of these values

VisitExp: Evaluate expression, taking values from lattice cells where defined

If value of \( e \) changes,
- if its an asst., add all SSA edges starting at asst. to SSAWL
- if its a pred., and value is \( \perp \), then add all exit edges to FlowWL
- value is a constant, add appropriate edge to Flow WL

Sparse Simple CP Example

```
A := 2
B := 3
A < B
C1 := 4
C2 := 5
C3 := \phi(C1, C2)
```

Entry

\[ SSAWL \]

FlowWL

Exit