**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)
write(A+3)
t
D = B + 2

E := A + B
write(E)

Exit

*Benefit?*

---

**Lattice for Constant Propagation**

```
true ...c0 c1 c2 ... false
```

May be constant

```
constant
```

Not constant

```
any \[\sqcap\] = any
c_i \[\sqcap\] c_j = \begin{cases} 
  c_i & \text{if } i=j \\
  \bot & \text{if } i \neq j 
\end{cases}
```

Initial value:

```
\bot
```
Simple Constant Propagation (Kildall)

Propagate constants through CFG, each branch equally likely

Worklist algorithm

Worklist algorithm

\[ V_1 := V_2 + V_3 \]

\[ \text{IN} \quad V_1, V_2, V_3, \ldots, V_n \]

\[ \text{OUT} \quad l_1, l_2, l_3, \ldots, \text{IN} \]

\[ \text{OUT(V1)} = \text{eval}(l_2 + l_3) \]

\[ \text{Out(Vi)} = \text{In(Vi)} \text{ for } i \neq 1 \]

\[ \text{eval(l2+l3)} = \]

\[ \begin{aligned}
\text{IF both are constants} & \quad \text{Out(Vi)} \\
\text{Ow if l2 or l3 is} & \quad \text{Out(Vi)} \\
\text{Ow if l2 or l3 is} & \quad \text{Out(Vi)}
\end{aligned} \]

Simple CP Example

Entry

\[ A = 2 \]

\[ B = 3 \]

\[ A < B \]

\[ C := 4 \]

\[ C := 5 \]

\[ \text{Exit} \]
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 $e$

\[\begin{align*}
&\text{if not evaluated at compile time:} & \downarrow \\
&\text{no variables in } e & \text{eval(e)} \\
&\text{o.w.} & \uparrow \\
\end{align*}\]

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

\[\begin{align*}
V_1 := & \ V_1, V_2, V_3, \ldots, V_n \\
& \ l_1, l_2, l_3, \ldots, l_n \\
\end{align*}\]

\[\begin{align*}
e := & \ \text{Use}(V_1) \\
& \ V_1, V_2, V_3, \ldots, V_n \\
& \ j_1, j_2, j_3, \ldots, j_n \\
\end{align*}\]

Compute $n_1 = l_1 \cap j_1$

If $n_1 \neq j_1$, 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
 |
 v
 A = 2
 v
 B = 3
 v
 A < B
    |
    v
C1 = 4
    |
    v
C3 := phi(C1, C2)
    |
    v
 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 \geq 1
 v
 I > 1
 t
 J \geq 1
 v
 J := 2
 f
 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:

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

Conditional CP Example

Entry

A \triangleright 2

B \triangleright 3

A < B

C := 4

C := 5

\[ \text{Eval(Pred) \quad \begin{array}{l}
\text{then all branches exec.} \\
\text{true \quad then true branch exec.} \\
\text{false \quad then false branch exec.}
\end{array} \]

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( # Flow edges + # 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

otherwise do nothing
Sparse Conditional CP Algorithm, cont.

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

\[ \text{ex} \leftarrow \text{nex} \leftarrow \text{ex} \]

\[ V4 := \Phi(V1, V2, V3) \]

\[ \text{if, } \perp, \perp \]

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
  - if value is a constant, add appropriate edge to FlowWL