Dataflow Analysis Framework

Last week: Two Dataflow Analyses

Dataflow analysis
- Compute facts (live vars) over a control-flow graph (CFG)
  - Vertex(CFG) = statements
  - Edges(CFG) = program points
- Can build CFG for Joos/IR/assembly

Live Variable Analysis (LVA)

LVA: find set of vars live at each program point
(var is live at some program point := its current value may be needed)

in[n] = facts true on all in-edges (conservative estimation)
= vars that may be live before n executes

$in[n] = use[n] \cup (out[n] \setminus def[n])$ $F_n(l) = use[n] \cup (l \setminus def[n])$

out[n] = facts true on all out-edges (conservative estimation)
= vars that may be live after n executes

$out[n] = \bigcup_{n' > n} in[n']$ $\sqcap = \cup$

iterative solving of equations

$in[n] = use[n] \cup (\bigcup_{n' > n} in[n'] \setminus def[n])$

Algorithm: iterative solving

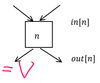
1. Initialize in[n] := ∅, for all n
2. Repeat until no change to in[•] is possible:
   For all n:
   $in[n] := use[n] \cup (\bigcup_{n' > n} in[n'] \setminus def[n])$

Inefficiency: has to perform update even for CFG nodes whose equations are currently satisfied.

Algorithm: worklist $\top = \emptyset$

1. Initialize in[n] := ∅, for all n
2. Set worklist (usually FIFO queue) w := all nodes in $Nodes$(CFG)
   Invariant: node n's equations are **not** currently satisfied ⇒ n ∈ w
3. While ∃ n ∈ w:
   $w := w \setminus \{n\}$
   $in[n] := use[n] \cup (\bigcup_{n' > n} in[n'] \setminus def[n])$
   If $in[n]$ changed, push predecessors of n onto w:
   $w := w \cup \{n' \mid n' < n\}$

Reachable Statement Analysis (RSA)



in[n] = facts true on all in-edges (conservative estimation)
= true/false: "program point before n may be reached"

- $in[n] = \bigvee_{n' < n} out[n']$ if in[n] has any predecessor $\sqcap = \vee$
- in[n] = false if in[n] does not have any predecessor and $n \neq$ Start
- in[Start] = true

out[n] = facts true on all out-edges (conservative estimation)
= true/false: "program point after n may be reached"

- out[return;] = false out[return e;] = false
- $out[n] = in[n]$

$F_n(l) = \begin{cases} false, & n = return \\ l, & otherwise \end{cases}$

Algorithm: iterative solving $\top = false$

1. Initialize out[n] := false, for all n
2. Repeat until no change to in[•] is possible:
   For all n:
   update $out[n]$ per its equation

Inefficiency: has to perform update even for CFG nodes whose equations are currently satisfied.

Algorithm: worklist

1. Initialize in[n] := false, for all n
2. Set worklist (usually FIFO queue) w := all nodes in $Nodes$(CFG)
   Invariant: node n's equations are **not** currently satisfied ⇒ n ∈ w
3. While ∃ n ∈ w:
   $w := w \setminus \{n\}$
   update $out[n]$ per its equation
   If $out[n]$ changed, push successors of n onto w:
   $w := w \cup \{n' \mid n' > n\}$

Available Copies Analysis

Goal: copy propagation.

```
x = y;
...
z = 3 * x;   // y is an avail. copy of x
...
x = q;
...
z = 3 * y
```

in[n] = equations that hold before n executes

$in[n] = \{ X_1 = y_1, \cdots X_n = y_n \}$ can replace x_i w/ y_i

$in[n] = \bigcap_{n' < n} out[n']$ $\sqcap = \cap$

CFG node	gen[n]	kill[n]
$x \leftarrow y$	$\{x=y\}$	$\{x=z \mid \forall z\} \cup \{z=x \mid \forall z\}$
$x \leftarrow e_{(\text{cfg. } \forall y)}$	$\emptyset$	$\{x=z \mid \forall z\} \cup \{z=y \mid \forall z\}$
$[e_1] \leftarrow e_2$	$\emptyset$	$\emptyset$
if e	$\emptyset$	$\emptyset$
return e	$\emptyset$	$\emptyset$
start	$\emptyset$	all possible copies

out[n] = equations that hold after n executes

$out[n] = gen[n] \cup (in[n] \setminus kill[n])$

$F_n(l) = gen[n] \cup (l \setminus kill[n])$

$\top$ = all possible copies

Worklist
1. Initialize $out[n] :=$ all possible copies
2. `~~~` worklist
3. While n ∈ worklist : ---

Dataflow analysis framework

Goal: compute $l \in L$ for each program point.

Five ingredients.

L = space of facts (aka dataflow values)

Top element $\top \in L$
maximal information
least conservative approx.

LVA: set of live vars
RSA: program point reachability
ACA: set of avail copies

D = direction ∈ { forward, backward }

Forward: $out[n] = F_n(in[n])$ (transfer function)

Backward: $in[n] = F_n(out[n])$

F_n transfer functions

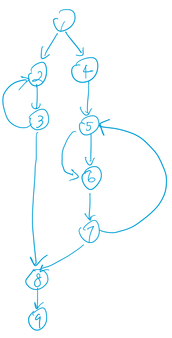
$\sqcap$ meet operator conservatively combines facts

Forward: $in[n] = \bigsqcap_{n' < n} out[n']$

Backward: $out[n] = \bigsqcap_{n' > n} in[n']$

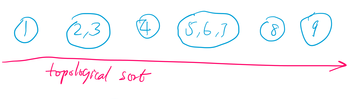
Forward / Backward
Init $out[n] = \top \ \forall n$
Repeat until convergence:
For all nodes:
$out[n] = F_n(\bigsqcap_{n' < n} out[n'])$
$in[n] = F_n(\bigsqcap_{n' > n} in[n'])$

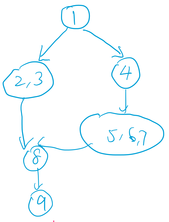
Worklist
Init $out[n] = \top, \forall n$
Init w = all nodes
While ∃ n ∈ w
$w := w \setminus \{n\}$
$in[n] = F_n(\bigsqcap_{n' > n} out[n'])$
if $out[n]$ changed
$w := w \cup \{w' \mid w' < n\}$



F_n

Find SCCs and create DAG of SCC





Apply worklist algorithm to each SCC
propagate information through DAG

Big Reveal

N nodes in CFG.

Start from $(\top, \cdots, \top) \in L^N$

Results is $(l_{n_1}, \cdots, l_{n_N}) \in L^N$

Global transfer function $F: L^N \to L^N$

$$F(l_{n_1}, \cdots, l_{n_N}) = \left(F_{n_1}\left(\bigsqcap_{n' < n_1} l_{n'}\right), \cdots, F_{n_N}\left(\bigsqcap_{n' < n_N} l_{n'}\right) \right)$$

Iterative Solving.
Initialize $X = (\top, \cdots, \top) \in L^N$
Repeat until convergence:

$$X = F(X)$$

Final X is a fixed point of F i.e. $X = F(X)$