Live Definitions Problem

Def: A def D of X is ***live*** at node N if there is a path p from N to Exit with a use of X that can use the value defined at D. Otherwise, the def is ***dead.***

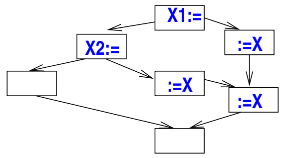
Def: The variable X is ***live*** at node N if there is def D of X that is live at N.

Live Variables Problem: What variables are live at N?

Useful for storage reuse, register allocation

Liveout(B) = set of variables live on exit from node B

Ex:



Solving Live Variables Problem

Local Analysis:

Gen(B): set of variables used in B

Pres(B): set of variables preserved by B

Data Flow Equations:

Initialize to ϕ

$$Livein(B) = \bigcup_{S \text{ succ of } B} Liveout(S)$$

$$Liveout(B) = Gen(B) \cup (Livein(B) \cap Pres(B))$$

Solution Method: Iterate until no change in Live sets.

Most optimistic: ϕ *Most pessimistic: all defs*

What is common?

Flow graph: *CFG*

Local Analysis: *Gen, Pres*

Meet Function: $\cup$

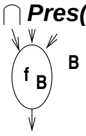
Direction in flow graph: *forward, backward*

Data Flow Equations:

$$In(B) = \bigcup_{S \text{ succ of } B} Out(S)$$

$$Out(B) = Gen(B) \cup (In(B) \cap Pres(B))$$

General case function:



$$Out(B) = (f_B\ In(B))$$

General Data Flow Framework

Flow Graph $\mathcal{F}$ Ex CFG, Reverse CFG, call graph

Lattice $\mathcal{L}$

Domain $\mathcal{D}$ Ex all sets of defs, all sets of uses

Meet operator $\sqcap$ Ex $\cup$, $\cap$

commutative, associative,...

Special elements $\top$, $\perp$ Ex all sets of defs, ϕ

Top, Bottom

$X \sqcap \perp = \perp$ $X \sqcap \top = X$

Induced partial order $X \leq Y$ iff $X \sqcap Y = X$ Ex. $\subseteq$

Seto of Transfer functions $f : \mathcal{D} \rightarrow \mathcal{D}$

Assigned to nodes

include id, constant functions, closed under $\circ$

Iterative Algorithm

Initialization: Reverse CFG if backward problem.
Initialize ***Out(B), In(B)***

Worklist Algorithm:

Worklist := Set of all nodes

While(***Worklist*** $\neq$ ***0***)

Remove node ***N*** from worklist

OldOut := Out(N)

$In(B) = \sqcap_{P\ pred\ of\ B}\ Out(P)$

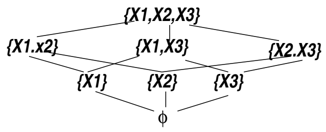
$Out(B) = f_B(In(B))$

if (***Out(N)*** $\neq$ ***OldOut***) then add successors of ***N*** to ***Worklist***

How know iterative algorithm works?

Termination:

Finite Lattice Functions have finite no. of possible values



Monotone Functions are non-increasing or decreasing

Solution:

Iterate till no change: All equations simultaneously satisfied

Quality of solution?

Quality of Iterative Solution

Best Solution: Holds for all ***real*** paths taken during program execution

Meet-Over-all Paths (MOP):

Iterative solution over all paths

$$MOP(B) = \bigsqcap_{\text{all paths } p \text{ from Entry to } B} f_p(\top) \le Best$$

where $\top$ *is Top*

and f_p *is* $f_{B1} \circ f_{B2} \circ \ldots \circ f_{Bn}$

for $p = B1 \ \ldots \ Bn$.

MOP is undecidable (even if monotonic)

When can MOP be achieved?

Distributive property: $f(X \sqcap Y) = f(X) \sqcap f(Y)$

Merge then apply f same as apply f then merge

Distributive $\longrightarrow$ **MOP solution via iterative algorithm**

What is not distributive?

Constant propagation

Ex. $X := 2$, $Y := 3$ $\qquad$ $X := 3$, $Y := 2$

$\qquad := X + Y$

Maximal Fixed Point (MFP) Solution:

Achieved by iterative algorithm on all problems covered.

MFP = MOP if distributive

MFP $\le$ MOP $\le$ Best

Summary Iterative Algorithm

Complexity: O(N^2) where N = size of FG

May take long to converge

Can improve by good choice of node order, ...

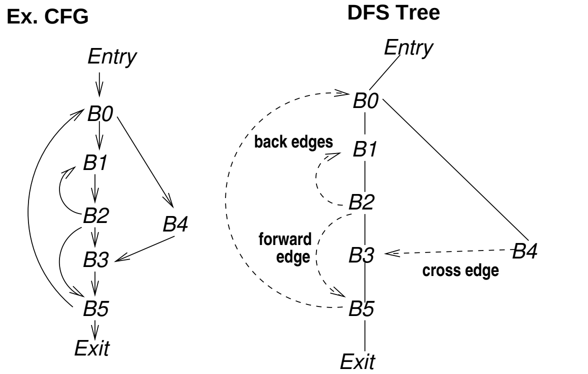
Simple to implement

Handles irreducible graphs

Doesn't recognize program structure

Loops, intervals

Constructing Tarjan Intervals



Pre-order: process node before descendants

(Post-order: after desc.)

Reverse Pre-order

Constructing Tarjan Intervals (cont’d)

Back:

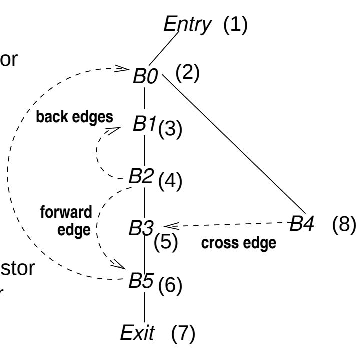
From node to an ancestor in tree

Forward:

From node to a descendant in tree

Cross:

From node to non-ancestor with smaller DF number



Qu: How tell if ancestor or descendant?

Constructing Tarjan Intervals (cont’d)

For each back edge B → H, in reverse pre-order of headers:

Find Interval I(H):

Starting from B, traverse CFG edges backwards.

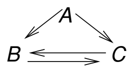
Stop when get to header H, or node with lower number than H.
(in the latter case, the CFG is irreducible).

The nodes traversed (including H) constitute I(H).

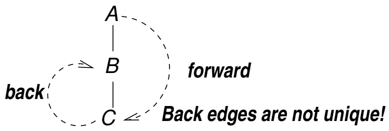
Replace nodes in I(H) with new node $N_{I(H)}$ that points to it.

Result: Hierarchical CFG with Tarjan Intervals

Irreducible subgraph



DFS tree



Irreducible subraph is an interval!

Interval-based Data Flow Analysis

Local Propagation:

For each interval, in order inner to outer,
- collect local info for each node in the interval
- use the local info for the interval nodes to collect info for the node representing the interval

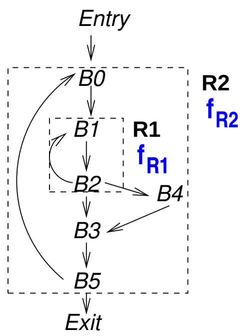
Global Propagation:

For each interval, in order outer to inner,
- given the IN set for the header to the interval,
- propagate global info to all nodes in the interval

Result: CFG with global data flow info

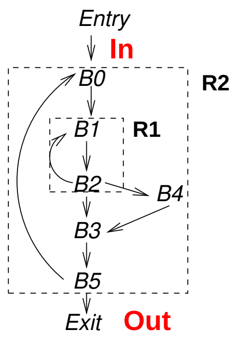
Interval-based Solution

LOCAL: inner to outer



$$f_{R1} = \lim_{n} (f_{B2} \circ f_{B1})^n$$

GLOBAL: outer to inner



$$f_{R2}(In) = Out$$

Reaching Defs Example

	Gen	Kill
(1) L1:Read(N)	N1	N2
(2) Call FOO(N)	N2	
(3) I := 1	I3	I6
(4) Repeat Until(I > N)		
(5) A(I):= A(I) + 1	A5	
(6) I := I + 1	I6	I3
(7) Endrepeat		
(8) Print(A(N))		
(9) Goto L1		

Local: *Gen(n), Pres(n) for each node n in I*

Global: $Out(h) = Gen(h) \cup (In(I) \cap Pres(h))$ *for header h*

$$In(n) = \bigcup_{p \text{ pred of } n} Out(p)$$

$$Out(n) = Gen(n) \cup (In(n) \cap Pres(n))$$

Summary Interval-based Algorithm

Complexity: O(E α (E)) ***if reducible*** (α (E) grows slowly)
exponential ***if irreducible***

Backward Problems more difficult
Reverse CFG can have multi-entry loops

More complicated implementation

Uses more space

Irreducible subgraphs handled separately

Allows for incremental update

Often used in practice