Data Flow Analysis

Static (compile-time) analysis of how data *flows* during execution

Undecidable in general for *real* execution

Prove small facts about program

Determine legality of specific optimizations

Need conservative answers

Want most optimistic solution

Data Flow Information

```
Ex. (S1)   Y := 1
    (S2)   X := K
    (S3)   T := FOO(Z)* Y + FOO(Z)
    (S4)   Y := Y + K
```

Determine if X a constant in loop

Determine if FOO modifies Z

Determine if expression FOO(Z)* Y + FOO(Z) already computed

Determine if Y not used later in program

Reaching Definitions Problem

Def: A ***definition*** of a variable is a possible assignment to the variable. We say that the statement containing the definition ***defines*** the variable.

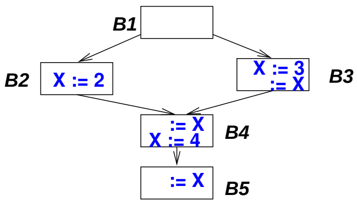
Def: If a definition ***always*** assigns to the variable, it ***kills*** all other definitions. Otherwise, it ***preserves*** them.

Ex. X := A is a definition, and kills all other defs. of X
FOO(Z) may define X but preserves all defs. of X

Def: A definition D of X at node B1 ***reaches*** node B2 if there is a path p from B1 to B2 such that D is preserved on path p.

Reachin(B) = set of defs that reach the entry of node B

Reaching Definitions Example



Def of X at node B2 reaches B4 (not B1, B3, B5)

Def of X at node B4 reaches B5 and kills defs at B2, B3

Useful for constant propagation, code motion

Solving Reaching Definitions Problem

Local Analysis:

Gen(B): set of local defs that reach the end of B

Pres(B): set of defs preserved by B

Data Flow Equations:

Reachin(B): set of defs that reach entry of B

Reachout(B): set of defs that reach exit of B

Initialize to ϕ (empty set)

$$Reachin(B) = \bigcup_{P \text{ pred of } B} Reachout(P)$$

$$Reachout(B) = Gen(B) \cup (Reachin(B) \cap Pres(B))$$

Solution Method: Iterate until no change in Reach sets. (fixed point)

Most optimistic: ϕ *Most pessimistic: all defs (satisfies the equations!)*

Reaching Defs Example

	Gen	*Pres*	*Reachin*	*Reachout*
(1) L1:Read(N)	N1			
(2) Call FOO(N)	N2	N1		
(3) I := 1	I3			
(4) Repeat Until(I > N)				
(5) A(I):= A(I) + 1	A5			
(6) I := I + 1	I5			
(7) Endrepeat				
(8) Print(A(N))				
(9) Goto L1				

With sets or bit vectors

Takes time to propagate info , entire pass with no change

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.***

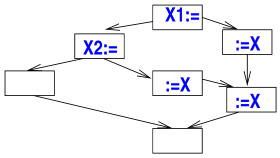
Def: The variable X is ***live*** at node N if there is def D of X that is live at N.

Live Variables Problem: What variables are live at N?

Useful for storage reuse, register allocation

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

Ex:



Solving Live Definitions Problem

Local Analysis:

Gen(B): set of local uses in B

Pres(B): set of defs 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*

Other Data Flow Problems

Available Expressions:

An expression e is ***available*** at B if every path to B contains a computation of e from defs that are live at B.

Forward, 1 bit per expression

Upwards Exposed Uses:

Set of uses that may not be defined.

Backward, 1 bit per expression

Partially Redundant Expressions:

Set of expressions appearing at least twice on some path, without its operands being modified between occurrences of the expression.

Bidirectional, 1 bit per expression

What is common?

Flow graph: *CFG*

Local Analysis: *Gen, Pres*

Meet Function: $\cup$

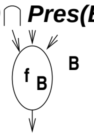
Direction in flow graph: *forward, backward*

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

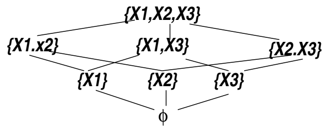
Out(B) $= f_B(In(B))$

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

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

where $\top$ ***is Top***

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

for p = B1 ... 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.

Left predecessor	Right predecessor
$X := 2$	$X := 3$
$Y := 3$	$Y := 2$

$:= X + Y$

Maximal Fixed Point (MFP) Solution:

Achieved by iterative algorithm on all problems covered.

MFP = MOP if distributive

MFP $\leq$ MOP $\leq$ 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