Data Flow Analysis and Optimizations

Last Time

- Control Flow Graphs

Today

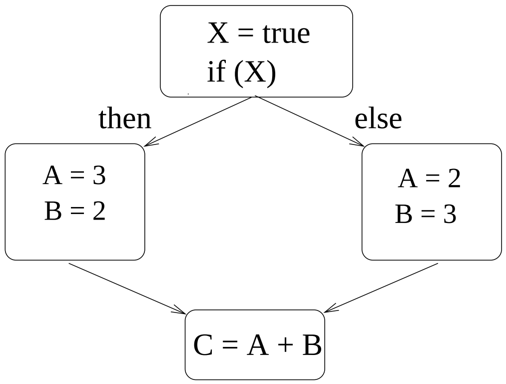
- Data Flow Analysis
- Data Flow Frameworks
- Constant Propagation
- Reaching Definitions

Data Flow Analysis

Data flow analysis tells us things we want to know about programs, for example:

- Is this computation loop invariant?
- Which definition reaches this use?
- Is this value a constant?

Example:



Data Flow Analysis

Systems of equations that compute information (e.g., uses, definitions, values) about variables at program points.

A **Monotone Data Flow Framework**

- *point* - start and/or end of a basic block
- Information for a forward problem
  $\text{INFO}_{in}(v) \;=\;$ merge $(\text{INFO}_{out}\text{predecessors}(v))$
  $\text{INFO}_{out}(v) \;=\;$ transfer $(\text{INFO}_{in}(v))$
- Transfer functions:
  T_v is the transfer function for v, how information is changed by v.
  T_q is the transfer function for a path and describes how information is carried on path q. All paths start at the entry *entry*.

Given Q: *entry* $\stackrel{+}{\rightarrow}$ x, where x is a node in the CFG, such that $Q = q_o \rightarrow q_1 \rightarrow \ldots q_n$, the **transfer function** is:

$$t_{q_n-1}(t_{q_n-2}(\ldots(t_2(t_1(t_0(\top)))\ldots)$$

Meet Over All Paths Solution

$$mop(x) = \sqcap_{Q \in Paths(x)} t_Q(\top)$$

Data Flow Framework

1. A *semilattice* $\mathcal{L}$ with a binary meet operation $\sqcap$, such that $a, b, c \in \mathcal{L}$:
   - $a \sqcap a = a$ (*idempotent*)
   - $a \sqcap b = b \sqcap a$ (*commutative*)
   - $a \sqcap (b \sqcap c) = (a \sqcap b) \sqcap c$ (*associative*)
2. $\sqcap$ imposes an order on L, $\forall\ a, b \in L$
   - $a \succeq b \Leftrightarrow a \sqcap b = b$
   - $a \succ b \Leftrightarrow a \succeq b$ and $a \neq b$
3. A semilattice has a *bottom* element $\perp$, *iff*
   - $a \sqcap \perp = \perp$ for every $a \in \mathcal{L}$.
   - $\forall a \in \mathcal{L}, a \succeq \perp$
4. It has a *top* or *identity* element, *iff*
   - $a \sqcap \top = a$ for every $a \in \mathcal{L}$
   - $\forall a \in \mathcal{L}, \top \succeq a$

Problem Representation

- choose a semilattice L to represent facts
- attach to each $a \in L$ a *meaning*
  each $a \in L$ is a distinct a set of known facts
- with each node n, associate a function $f_n : L \rightarrow L$
  f_n models behavior of code corresponding to n
- let $\mathcal{F}$ be the set of all functions the code generates

Constant Propagation Example Framework

Constant propagation lattice: $\top$

... -2 -1 0 1 2 ...

$\bot$

1. meet rules
   - a ⊓ ⊤ = a
   - a ⊓ ⊥ = ⊥
   - constant ⊓ constant = constant (if equal)
   - constant ⊓ constant = ⊥ (if not equal)
2. meet properties impose a partial order on L
   - 3 ⊓ 3 = 3
   - 3 ⊓ 2 = 2 ⊓ 3
   - 3 ⊓ (2 ⊓ 4) = (3 ⊓ 2) ⊓ 4
3. bottom
   - $a \sqcap \bot = \bot$ for every $a \in \mathcal{L}$.
   - $\forall a \in \mathcal{L}, a \succeq \bot$
4. top
   - $a \sqcap \top = a$ for every $a \in \mathcal{L}$
   - $\forall a \in \mathcal{L}, \top \succeq a$

Data Flow Framework

A descending chain in L is a sequence $x_1, x_2, \cdots, x_n$,

a) $1 \leq i \leq n, x_i \in L$, and

b) $1 \leq i < n, x_i \leq x_{i+1}$

If $\forall a \in L$, $\exists$ constant b_a such that any chain beginning with a has length $\leq b_a$, we say that L is bounded.

Any bounded semilattice has *finite descending chains*.

Admissible Function Spaces [Kam & Ullman]

For a bounded semilattice L, $\mathcal{F} : L \rightarrow L$ is an *admissible function space iff*:

1. Monotonic:

$$\forall\ f \in \mathcal{F}, \forall\ x, y \in L,\ x \preceq y \Longrightarrow f(x) \preceq f(y)$$

2. Identity operation:

$$\exists f_i \in \mathcal{F},\ \text{such that}\ \forall\ x \in L,\ f_i(x) = x$$

3. Closed under composition:

$$f, g \in \mathcal{F} \Rightarrow f \circ g \in \mathcal{F}, \text{where}\ \forall\ x \in L,\ [f \circ g](x) = f(g(x))$$

4. $\perp$ exists to any $x \in \mathcal{L}$

$$\forall\ x \in L,\quad \exists\ \text{a finite subset}\ H \subseteq \mathcal{F}\quad \ni x = \sqcap_{f \in H} f(0)$$

Monotone Data Flow Framework

is a triple $\langle \mathcal{L}, \sqcap, \mathcal{F} \rangle$ where

- $\sqcap$ is the meet operation, or *confluence* operator.
- $\langle \mathcal{L}, \sqcap, \rangle$ is a semilattice of finite length with bottom $\bot$.
- $\mathcal{F}$ is a *monotone operation space* on $\mathcal{L}$

A monotone operation space on a semilattice $\langle \mathcal{L}, \sqcap, \rangle$ is a set of unary functions such that for each operation $f \in \mathcal{F}$ is monotonic:

$$(\forall f \in \mathcal{F})(\forall x, y \in \mathcal{L})[f(x \sqcap y) \preceq f(x) \sqcap f(y)]$$

A **Distributive** framework

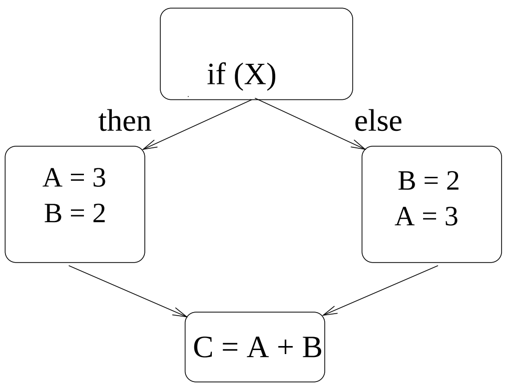
$$(\forall f \in \mathcal{F})(\forall x, y \in \mathcal{L}) f(x \sqcap y) = f(x) \sqcap f(y)$$

Meet Over All Paths Solution

$$mop(x) = \sqcap_{Q \in Paths(x)} t_Q(\top)$$

Constant Propagation Example Framework

1. Is CP monotonic?
2. Is CP distributive?
3. Is every solution a meet overall paths solution?



Reaching Definitions

For each vertex, find the set of variable definitions that might reach that vertex.

GEN(v) - variable v *may* be defined or assigned to

KILL(v) - variable v is defined, overwriting other definitions

```
1:  read N
2:  call check (N)
3:  I = 1
4:  while (I < N) do
5:     A(I) = A(I) + I
6:     I = I + 1
7:  endwhile
8:  print A(N)
```

	GEN	KILL	PRED	SUCC
1:	N_1	N		2
2:	N_2		1	3
3:	I_3	I	2	4
4:			3,7	5,8
5:	A_5		4	6
6:	I_6	I	5	7
7:			6	4
8:			4	

Reaching Definitions - Transfer Function

IN(v) - the set of definitions that reach statement v

$$\mathrm{IN}(v) = \bigcup_{p \in \mathrm{PRED}(v)} \mathrm{OUT}(p)$$

OUT(v) - the set of reaching definitions just after statement v

$$\mathrm{OUT}(v) = \mathrm{GEN}(v) \cup (\mathrm{IN}(v) - \mathrm{KILL}(v))$$

- IN is an *inherited* attribute;
- OUT is a *synthesized* attribute;
- GEN and KILL are *basic* attributes.
- *Forward* data flow problems propagate information from predecessors of a vertex v to v.

  Backward data flow problems propagate from successors of v to v.

Reaching Definitions

Monotone Data Flow Framework

- A = set of *generations*,
  generation = (statement, variable)
- Lattice: $\mathcal{L}$ = $\langle$ powerset(A), $\cup$ - set union $\rangle$
  powerset(A) is the set of all subsets of A
  What does it look like?
- initial value = $\emptyset$
- transfer function T_v: $T_v(x) = (x - \text{KILL}(v)) \cup \text{GEN}(v)$
- monotone: $x \leq y \;\Rightarrow\; T_v(x) \leq T_v(y)$
- distributive: $T_v(x \cup y) = T_v(x) \cup T_v(y)$

Work List Iterative Algorithm

initialize ReachingDefinitions(n)
worklist $\leftarrow$ the set of all nodes
while (*worklist* $\neq \emptyset$)
    pick and remove a node n from *worklist*
    recompute ReachingDefinitions(n)
    if ReachingDefinitions(n) changed then
        worklist $\leftarrow$ *worklist* $\cup$ SUCC(n)

initialization
IN(v) =
OUT(v) =

computation
IN(v) =
OUT(v) =

Reaching Definitions Algorithm

for $v \in V$
    IN(v) = $\emptyset$
    OUT(v) = GEN(v)
endfor
worklist $\leftarrow v \in V$
while (*worklist* $\neq \emptyset$)
    pick and remove a node v from *worklist*
    IN(v) = $\bigcup$ (OUT(p)), $p \in$ PRED(v)
    OUT(v) = GEN(v) $\bigcup$ (IN(v) - KILL(v))
    if OUT(v) changed then
        worklist $\leftarrow$ *worklist* $\cup$ SUCC(v)
endwhile

Reaching Definitions

For each vertex, find the set of variable definitions that might reach that vertex.

GEN(v) - variable v *may* be defined or assigned to

KILL(v) - variable v is defined, overwriting other definitions

1: read N
2: call check (N)
3: I = 1
4: while (I < N) do
5:     A(I) = A(I) + I
6:     I = I + 1
7: endwhile
8: print A(N)

	GEN	KILL	PRED	SUCC
1:	N_1	N		2
2:	N_2		1	3
3:	I_3	I	2	4
4:			3,7	5,8
5:	A_5		4	6
6:	I_6	I	5	7
7:			6	4
8:			4	

Reaching Definitions Example

	Initial value		iteration 1		iteration 2		iteration 3	
	IN	OUT	IN	OUT	IN	OUT	IN	OUT
1								
2								
3								
4								
5								
6								
7								
8								

Next Time

Questions for the Reaching Definitions Algorithm

- Does this always terminate?
- What answer does it compute?
- How fast (or slow) is it?