Reaching Definitions (from PPA, pp. 5)

An assignment (definition) of the form $[x := a]^l$ *may reach* a program point l', if

- there is an execution of the program that reaches l' where $\mathtt{x}$ was last assigned a value at l

Observation

Note the emphasis of *may reach*

It is very common to distinguish *may* analyses from *must* analyses.

A helpful way to think about this distinction is to think in terms of program paths.

- *may* X means there exists a path on which X happens
- *must* X means for all paths X happens

What if we have an analysis problem that requires X on no paths?

Reaching Definitions for Example

```
[y := x]^1;
[z := 1]^2;
while [y>1]^3 do
(
  [z := z*y]^4;
  [y := y-1]^5
);
[y := 0]^6
```

Clearly the definition $[y := x]^1$ may reach $[z := 1]^2$, we describe this more compactly by saying $(y, 1)$ reaches the entry 2.

Reaching Definitions for Example

$\{(x,?),(y,?),(z,?)\}$

```
[y := x]^1;
[z := 1]^2;
while
  [y>1]^3
  do (
    [z := z*y]^4;
    [y := y-1]^5
);
[y := 0]^6
```

Reaching Definitions for Example

$\{(x,?),(y,?),(z,?)\}$	`[y := x]`^{1}`;`	$\{(x,?),(y,1),(z,?)\}$
$\{(x,?),(y,1),(z,?)\}$	`[z := 1]`^{2}`;`	$\{(x,?),(y,1),(z,2)\}$
	`while`	
$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$	`[y>1]`^{3}	$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$
	`do (`	
$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$	`[z := z*y]`^{4}`;`	$\{(x,?),(y,1),(y,5),(z,4)\}$
$\{(x,?),(y,1),(y,5),(z,4)\}$	`[y := y-1]`^{5}	$\{(x,?),(y,5),(z,4)\}$
	`);`	
$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$	`[y := 0]`^{6}	$\{(x,?),(y,6),(z,2),(z,4)\}$

Denoting Reaching Definitions

We can describe the set of reaching definitions for the entry and exit of each program point as a pair of functions

$(\mathsf{RD}_{entry}, \mathsf{RD}_{exit})$

where

$\mathsf{RD}_i : \mathsf{Lab}_* \rightarrow \mathcal{P}(\mathsf{Var}_* \times (\{?\} \cup \mathsf{Lab}_*))$

Notes

- Lab_* and Var_* are the subsets of labels and variables occurring in the program under analysis, e.g., $\{1, \ldots, 6\}$ and $\{x, y, z\}$
- $?$ means that the origin of a definition is unknown

Reaching Definitions for Example

l	$\text{RD}_{entry}(l)$	$\text{RD}_{exit}(l)$
1	{ (x,?), (y,?), (z,?) }	{ (x,?), (y,1), (z,?) }
2	{ (x,?), (y,1), (z,?) }	{ (x,?), (y,1), (z,2) }
3	{ (x,?), (y,1), (y,5), (z,2), (z,4) }	{ (x,?), (y,1), (y,5), (z,2), (z,4) }
4	{ (x,?), (y,1), (y,5), (z,2), (z,4) }	{ (x,?), (y,1), (y,5), (z,4) }
5	{ (x,?), (y,1), (y,5), (z,4) }	{ (x,?), (y,5), (z,4) }
6	{ (x,?), (y,1), (y,5), (z,2), (z,4) }	{ (x,?), (y,6), (z,2), (z,4) }

Reaching Definitions for Example

For blocks that are assignments

- i.e., 1,2,4,5 and 6
- entry and exit values change for the defined variable

For blocks that are not assignments

- i.e., 3
- entry and exit values are the same

The effect of statements on the values are localized.

Usage of Reaching Definitions

Compiler Optimization

- An occurrence of a variable x at statement l is a constant c, if for all reaching definitions $(x, l') \in \mathsf{RD}_{entry}(l)$, the value assigned to x at l' is x.
- If all reaching definitions for an expression are outside the loop you can move the expression outside the loop (loop-invariant code motion)
- Construct program dependence graph (PDG). Edges in this graph for a statement l are built as follows: for each reaching definition $(x, l') \in \mathsf{RD}_{entry}(l)$, we introduce an edge (l', l).

Usage of Reaching Definitions

Software Engineering Tools

- If for a statement, l, that uses x there is a reaching definition $(x, ?) \in \text{RD}_{entry}(l)$ then there is a potential uninitialized use of x.
- Used in debugging: if a value at a breakpoint has a bad value then you set breakpoints at the reaching definitions and rerun.
- Program slicers are built using the PDG.

Safety of Reaching Definitions

Every reachable definition that the program can execute should be detected

May detect more definitions

- Identify fewer constants
- Fail to move some loop-invariant code
- Issue uninitialization warnings for initialized variables

Never miss a reaching definition

- Transform an expression based on mistaken impression it is constant
- Move expressions out of loop that change in loop
- Fail to issue a warning when variable may be uninitialized

Is it Safe?

Are the reaching definitions in the table safe for the example?

Do they contain all of the reaching definitions that can be executed?

Are there any *extra* reaching definitions?

Are these changes safe?

l	RD$_{entry}(l)$	RD$_{exit}(l)$
1	$\{$ (x,?), (y,?), (z,?) $\}$	$\{$ (x,?), (y,1), (z,?) $\}$
1’	$\{$ (x,?), (y,?), (z,?) $\}$	$\{$ (x,?), (y,?), (y,1), (z,?) $\}$
4	$\{$ (x,?), (y,1), (y,5), (z,2), (z,4) $\}$	$\{$ (x,?), (y,1), (y,5), (z,4) $\}$
4’	$\{$ (x,?), (y,1), (z,2), (z,4) $\}$	$\{$ (x,?), (y,1), (y,5), (z,4) $\}$

Data Flow Analysis

Traditionally, Data Flow Analyses exploit the results of *Control Flow Analysis*

> Information about the sequencing of execution of parts of a program's syntax.

For now we will assume the existence of relations that give a statement's predecessors $\mathsf{Pred} : \mathsf{Lab}_* \times \mathsf{Lab}_*$

- This is trivial to calculate for our *While* language
- Its easy to determine very approximate control flow info for Java
- Its hard to determine accurate info for Java and higher-order languages (e.g., where functions can be passed as arguments)

Often times Control Flow information is depicted in graphical form, e.g., as a flow chart.

Control Flow Info for Example

l	Pred(l)	Succ(l)
1	$\{$ 2 $\}$	$\{$ $\}$
2	$\{$ 3 $\}$	$\{$ 1 $\}$
3	$\{$ 4,6 $\}$	$\{$ 2,5 $\}$
4	$\{$ 5 $\}$	$\{$ 3 $\}$
5	$\{$ 3 $\}$	$\{$ 4 $\}$
6	$\{$ $\}$	$\{$ 3 $\}$

More approximate relations could be used here (e.g., the full product relation), but it would decrease the precision of the data flow analysis results.

Data Flow Analysis

We will formulate a Reaching Definitions flow analysis to calculate

- sets of facts, i.e., sets of pairs from $\text{Var}_* \times \text{Lab}_*$
- at entry and exit of each labeled statement

Note that we could choose to compute this information for other *points* in the program, e.g., inside the RHS of an assignment statement

Two Approaches to Formulating Data Flow Analyses

Equational approach

- Define a system of simultanous equations
- $2 \mid \mathsf{Lab}_* \mid$ variables (to record sets of facts)
- $2 \mid \mathsf{Lab}_* \mid$ equations (to capture effects of stmts)

Constraint approach

- Define a system of *inclusions*
- $2 \mid \mathsf{Lab}_* \mid$ variables (to record sets of facts)
- inclusion constraints expressing relationships between the sets of facts associated with different statements

The Equational Approach

For each labeled statement

- $\text{RD}_{entry}(l)$: set of reaching definitions data flow facts before execution of l
- $\text{RD}_{exit}(l)$: set of facts after execution of l

Statement's effects on data flow facts captured by equations

- $\text{RD}_{exit}(l) = f(\text{RD}_{entry}(l))$

The definition of f depends on the kind of statement

- $[\texttt{x := a}]^l$: $\text{RD}_{exit}(l) = \text{RD}_{entry}(l) \setminus \{(x, l') | l' \in \text{Lab}_*\} \cup \{(x, l)\}$
- $[\texttt{...}]^l$: $\text{RD}_{exit}(l) = \text{RD}_{entry}(l)$

Equations for statement effects

$$\mathsf{RD}_{exit}(1) = \mathsf{RD}_{entry}(1) \setminus \{(y,l')|l' \in \mathsf{Lab}_*\} \cup \{(y,1)\}$$

`[y := x]`^{1}

$$\mathsf{RD}_{exit}(2) = \mathsf{RD}_{entry}(2) \setminus \{(z,l')|l' \in \mathsf{Lab}_*\} \cup \{(z,2)\}$$

`[z := 1]`^{2}

$$\mathsf{RD}_{exit}(3) = \mathsf{RD}_{entry}(3)$$

`while [y>1]`^{3} `do`

$$\mathsf{RD}_{exit}(4) = \mathsf{RD}_{entry}(4) \setminus \{(z,l')|l' \in \mathsf{Lab}_*\} \cup \{(z,4)\}$$

`[z := z * y]`^{4}

$$\mathsf{RD}_{exit}(5) = \mathsf{RD}_{entry}(5) \setminus \{(y,l')|l' \in \mathsf{Lab}_*\} \cup \{(y,5)\}$$

`[y := y - 1]`^{5}

$$\mathsf{RD}_{exit}(6) = \mathsf{RD}_{entry}(6) \setminus \{(y,l')|l' \in \mathsf{Lab}_*\} \cup \{(y,6)\}$$

`[y := 0]`^{6}

The Equational Approach

Statement sequencing is accounted for by introducing equations that propagate facts between control flow predecessors and successors.

- $\mathsf{RD}_{entry}(l) = \bigcup_{l' \in \mathsf{Pred}(l)} \mathsf{RD}_{exit}(l')$

Since definitions are not introduced in transitioning between statements

- any fact that holds at the exit of a statement also holds at the entry of each of its successors

The initial statement has no predecessor so treated specially

- $\mathsf{RD}_{entry}(l_{init}) = \{(x, ?) | x \in \mathsf{Var}_*\}$

Equations for inter-statement propagation

$$
\begin{array}{rcl}
\mathsf{RD}_{entry}(1) & = & \{(x,?),(y,?).(z,?)\} \\
\mathsf{RD}_{entry}(2) & = & \mathsf{RD}_{exit}(1) \\
\mathsf{RD}_{entry}(3) & = & \mathsf{RD}_{exit}(2) \cup \mathsf{RD}_{exit}(5) \\
\mathsf{RD}_{entry}(4) & = & \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{entry}(5) & = & \mathsf{RD}_{exit}(4) \\
\mathsf{RD}_{entry}(6) & = & \mathsf{RD}_{exit}(3)
\end{array}
$$

Solving the system of equations

Two approaches

- Elimination/substitution method
- Fixpoint iteration method

We'll look at the second method

- Easily automated
- Finds the *least* solution
- Minor requirements on data flow facts and equations

Lattice of Data Flow Facts

Variables in the equations range over sets of data flow facts

- there are finitely many facts (21 for our problem)
- reaching definition facts are naturally ordered by inclusion

A powerset ordered by inclusion is a *complete lattice*

Posets

A *partially ordered* set (poset) is

- a set
- a partial order $\sqsubseteq$, which is a reflexive, transitive and anti-symmetric relation

While not guaranteed to exists, all of the posets we will discuss have greatest lower and least upper bounds.

Bounds

A subset, Y, of a poset has l as an

- *upper bound* if $\forall l' \in Y : l' \sqsubseteq l$
- *lower bound* if $\forall l' \in Y : l \sqsubseteq l'$

A *least upper bound* (lub) l of Y is an upper bound such that $l \sqsubseteq l'$ where l' is an upper bound of Y

A *greatest lower bound* (glb) l of Y is a lower bound such that $l' \sqsubseteq l$ where l' is a lower bound of Y

Finding Bounds

If they exist, lub and glb are unique.

The least upper bound of two subsets is denoted $l_1 \sqcup l_2$ which is called the *join* operator.

The greatest lower bound of two subsets is denoted $l_1 \sqcap l_2$ which is called the *meet* operator.

Complete Lattices

A *complete lattice* $L = (L, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ consists of

- a poset $(L, \sqsubseteq)$
- where lub and glb exist
- $\perp = \sqcap L$ is the *least element*
- $\top = \sqcup L$ is the *greatest element*

In practice we'll only require $\sqsubseteq$ and either $\sqcup$ and $\perp$ or $\sqcap$ and $\top$.

Sets of Reaching Definitions Facts

$\mathcal{P}(\mathsf{Var}_* \times (\{?\} \cup \mathsf{Lab}_*))$

- $\sqsubseteq = \subseteq$
- $\sqcup = \cup$
- $\perp = \emptyset$

So, we are working with a complete lattice

Vector Representation of Equations

Consider the 12 variables as a single vector value

$$\vec{\mathsf{RD}} = (\mathsf{RD}_{entry}(1), \mathsf{RD}_{exit}(1), \ldots \mathsf{RD}_{exit}(6))$$

12-tuples of complete lattice values, $\vec{\mathsf{RD}} = (\mathsf{RD}_1, \ldots, \mathsf{RD}_{12})$, form a complete lattice

- $\vec{\mathsf{RD}} \sqsubseteq \vec{\mathsf{RD}}'$ iff $\forall i \mathsf{RD}_i \subseteq \mathsf{RD}'_i$
- $\vec{\mathsf{RD}} \sqcup \vec{\mathsf{RD}}'$ iff $(\mathsf{RD}_1 \cup \mathsf{RD}'_1, \ldots, \mathsf{RD}_{12} \cup \mathsf{RD}'_{12})$
- $\perp = \vec{\emptyset} = (\emptyset, \ldots, \emptyset)$

Equations as a Function

We can write the equation system as

$$\vec{\mathsf{RD}} = F(\vec{\mathsf{RD}})$$

where

$$F(\vec{\mathsf{RD}}) = (F_{entry}(1)(\vec{\mathsf{RD}}), \ldots, F_{exit}(6)(\vec{\mathsf{RD}}))$$

where the component functions encode the equations, for example,

$$F_{exit}(2)(\ldots) = \mathsf{RD}_{entry}(2) \setminus \{(z, l')|l' \in \mathsf{Lab}_*\} \cup \{(z, 2)\}$$

Properties of Functions

We can be assured that solution methods will produce an answer to a system of equations if the function is monotone.

A function, $f : L \to L$, is *monotone* if

$$\forall l, l' \in L : l \sqsubseteq l' \Rightarrow f(l) \sqsubseteq f(l')$$

Such functions are also termed *order preserving* since a pair of values ordered in a poset/lattice will have their images under f in the same relative order

F is monotone

There are three forms of constituent functions in F, e.g.,

- $F_{entry}(2)(\ldots) = \mathsf{RD}_{exit}(1)$
- $F_{entry}(3)(\ldots) = \mathsf{RD}_{exit}(2) \cup \mathsf{RD}_{exit}(5)$
- $F_{exit}(2)(\ldots) = \mathsf{RD}_{entry}(2) \setminus \{(z, l') | l' \in \mathsf{Lab}_*\} \cup \{(z, 2)\}$

These are easily seen to be monotone in the lattice of reaching definition facts.

Fixed Point of F is a Solution

Given the monotonicity of F it follows that

- $\vec{\emptyset} \sqsubseteq F(\vec{\emptyset})$
- $F^n(\vec{\emptyset}) \sqsubseteq F^{n+1}(\vec{\emptyset})$

Given that the lattice is finite there must be some pair of iterates of F such that

- $F^n(\vec{\emptyset}) = F^{n+1}(\vec{\emptyset})$
- $F^n(\vec{\emptyset})$ is a *fixed point* of F
- $F^n(\vec{\emptyset})$ is a solution to the equation system

Least Solution

The fixed point of F is the least solution of the system of equations

Suppose $\vec{\text{RD}} = F(\vec{\text{RD}})$, i.e., $\vec{\text{RD}}$ is a fixed point

- $\vec{\emptyset} \sqsubseteq \vec{\text{RD}}$
- so, $F(\vec{\emptyset}) \sqsubseteq F(\vec{\text{RD}})$ (monotonicity)
- $F(\vec{\emptyset}) \sqsubseteq \vec{\text{RD}}$
- $F(F(\vec{\emptyset})) \sqsubseteq F(\vec{\text{RD}}) = \vec{\text{RD}}$ (monotonicity)
- ...
- $F^n(\vec{\emptyset})) \sqsubseteq F(\vec{\text{RD}}) = \vec{\text{RD}}$ (monotonicity)

Chaotic Iteration

The weakest specification of an iterative algorithm for calculating fix point solutions for flow analysis problems

$$
\begin{array}{l}
\mathsf{RD}_1 := \emptyset; \\
\ldots \\
\mathsf{RD}_{12} := \emptyset; \\
\texttt{while } \exists j : \mathsf{RD}_j \neq F_j(\mathsf{RD}_1, \ldots, \mathrm{RD}_{12}) \texttt{ do} \\
\texttt{(} \\
\quad \mathsf{RD}_j := F_j(\mathsf{RD}_1, \ldots, \mathsf{RD}_{12}) \\
\texttt{);}
\end{array}
$$

The algorithm continues to loop until all components have reached a local fixed point.

So, if the algorithm terminates a fixed point of F is reached.

Properties of Algorithm

Will it terminate?

- Executing the loop body means there is some $\mathsf{RD}_j \neq F_j(\mathsf{RD}_1, \ldots, \mathsf{RD}_{12})$
- this only happens if $\mathsf{RD}_j \subset F_j(\mathsf{RD}_1, \ldots, \mathsf{RD}_{12})$
- so, the assignment in the loop body *increases* the size of RD_j
- but, this can only happen a finite number of times since the lattice is finite

Will it produce the least solution?

Think about that the fixed point solution is the least solution and apply the hint from problem 1.6.