Lecture 1: Compiler Fundamentals

Patrick Lam

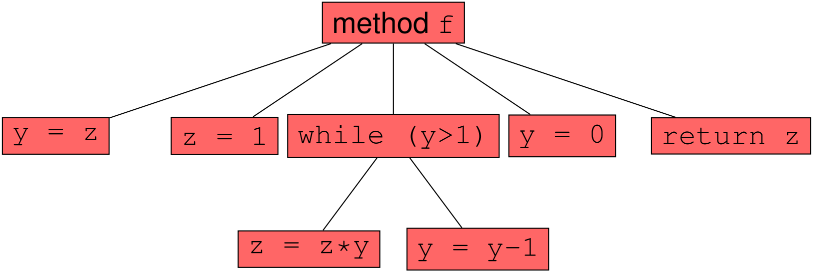
September 12, 2008

Example Method

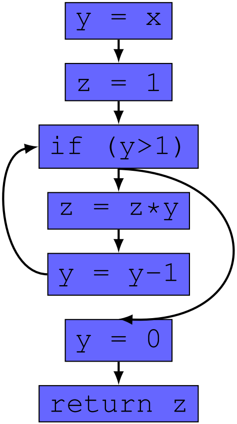
```
public int f(int x) {
  int y, z;

  y = x;
  z = 1;
  while (y > 1) {
    z = z * y;
    y = y - 1;
  }
  y = 0;
  return z;
}
```

Abstract Syntax Tree



Control Flow Graph

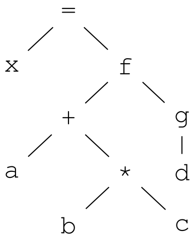


Three-Address Code

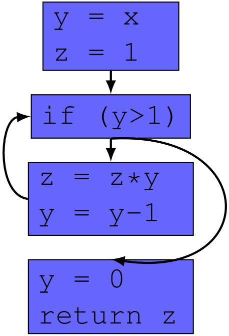
```
x = f(a+b*c, g(d))
```

3-Address Code vs. Expression Tree

```
t0 = b*c
t1 = a+t0
t2 = g(d)
x = f(t1, t2)
```



CFG on Basic Blocks



Basic Block Definition

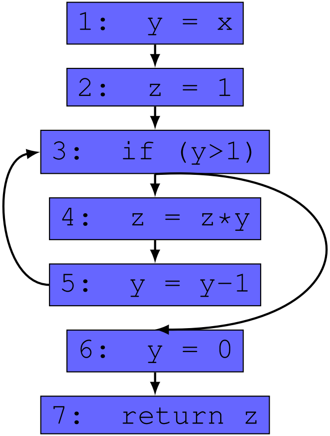
A basic block is a sequence of statements with no jumps into or out of the block.

Dataflow Analysis in Brief

Set up an abstract domain and calculate the effect of each program statement with respect to your abstract domain, until the fixed point.

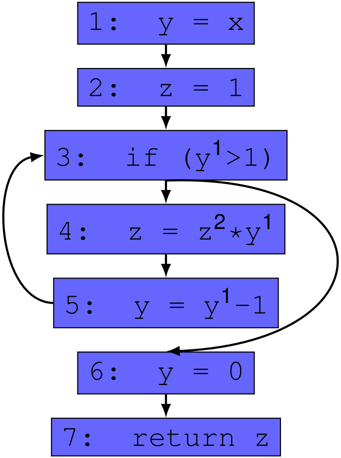
Analysis 1: Reaching Definitions

Q: Which definitions reach a given variable use?



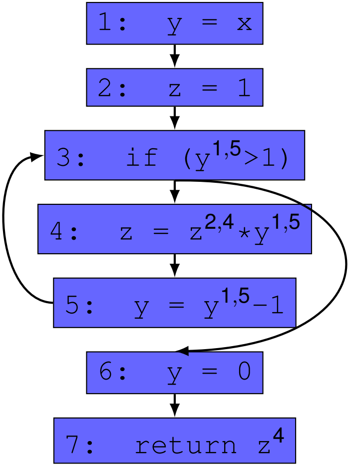
Analysis 1: Reaching Definitions

Q: Which definitions reach a given variable use?



Analysis 1: Reaching Definitions

Q: Which definitions reach a given variable use?



Setting up a Dataflow Analysis

Six steps:

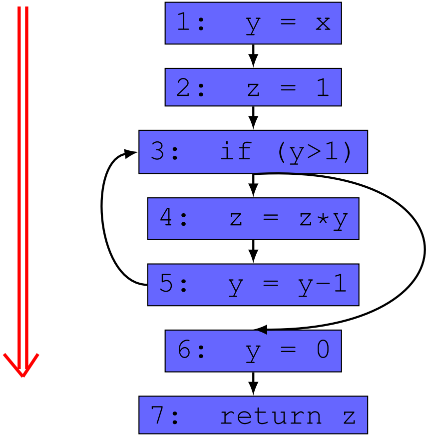
1. What is your problem?
2. Forward or backward?
3. What's in your dataflow sets?
4. Merge: union or intersection?
5. What are your transfer functions?
6. What are the initial values?

Reaching Definitions Problem Statement

A definition d of variable v reaches a use u if there exists a path of control-flow edges from d to u that does not contain any redefinitions of v.

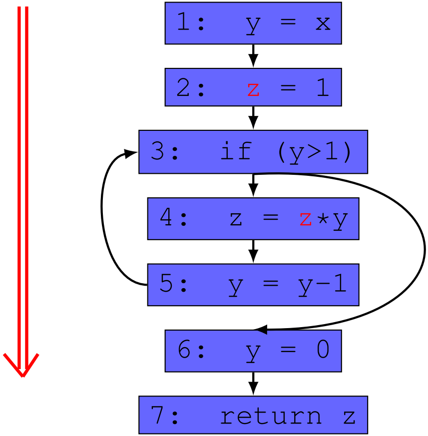
Reaching Definitions: A Forward Analysis

We move information forward through the CFG.



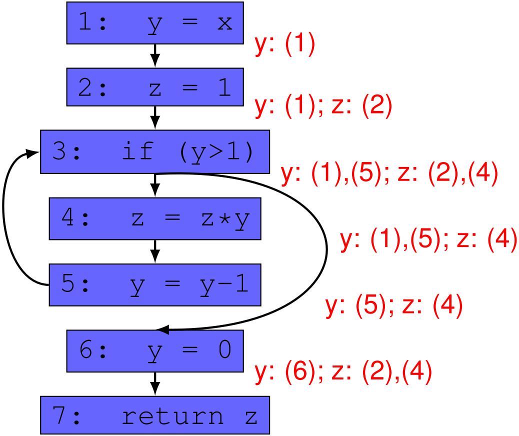
Reaching Definitions: A Forward Analysis

We move information forward through the CFG.



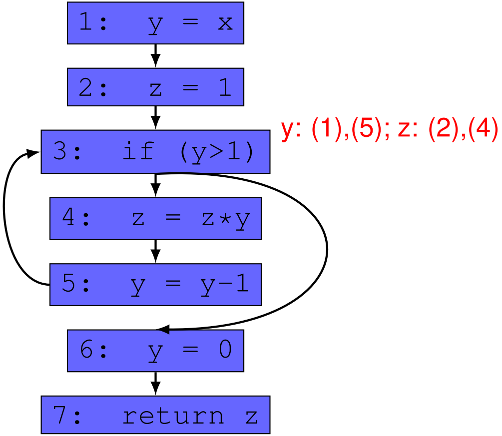
Reaching Definitions: Abstraction

Keep a list of definitions for each variable.



Reaching Definitions: Merge Operator

A definition reaches if any path exists from def to use: union.



Reaching Definitions: Transfer Functions

For a some-paths analysis like reaching definitions:

$$\mathrm{out}(s) = \left(\bigcup_{i \in \mathrm{preds}(s)} \mathrm{out}(i) - \mathrm{kill}(s) \right) \cup \mathrm{gen}(s)$$

Reaching Definitions: Kill Sets

At an assignment statement,

$$s : v = \mathrm{RHS},$$

we kill all extant definitions for variable v:

$$v : *.$$

Reaching Definitions: Gen Sets

At an assignment statement,

$$s : v = \text{RHS},$$

we generate a new definition s for variable v:

$$v : s.$$

Reaching Definitions: Initial Values

- At the beginning of the procedure, no definitions reach any variables; for all variables v, we use $\emptyset$.
- At all other program points p, we start by assuming that no definitions reach p either: also use $\emptyset$.

Dataflow Analysis Discussion

Six steps:

1. What is your problem?
2. Forward or backward?
3. What's in your dataflow sets?
4. Merge: union or intersection?
5. What are your transfer functions?
6. What are the initial values?

Forward Analysis: Reaching Defs

```
public int f(int x) {
  int y, z;

  y = x;
↓ z = 1;
  while (y > 1) {
    z = z * y;
    y = y - 1;
  }
  y = 0;
  return z;
}
```

Backward Analysis: Live Locals

```
  public int f(int x) {
    int y, z;

    y = x;                          // { y }
    z = 1;                          // { y, z }
    while (y > 1) {
      z = z * y;                    // { z }
      y = y - 1;                    // { y, z }
    }
    y = 0;                          // { z }
    return z;
↑ }
```

Dataflow Sets

Some more examples:

- "x points to `null` / non-`null` / don't-know"
- "y is positive / negative / zero / don't-know"

$\perp$ ("bottom") represents "don't know"—no information yet.

$\top$ ("top") represents "overdetermined"—e.g. our analysis wants one precise answer (e.g. constant propagation: $y = 5$) and we have more than one answer ($y = 5$ *and* $y = 3$).

Merge operator

$\bigcup$: some-path analysis
(e.g. reaching defs)

$\bigcap$: all-paths analysis
(e.g. available expressions)

Transfer functions

Assignment statements and invoke statements are most interesting.

Initial Values

At entry point, need conservative underapproximation.

At other points, use overapproximation; it gets refined later.

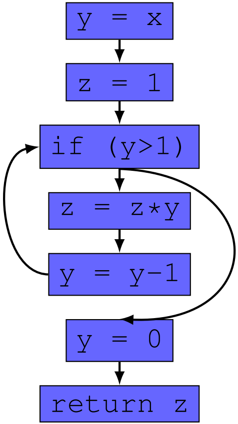
Beware

Dataflow analysis is subtle!

Alternatives

- Constraint Systems
- Type and Effect Systems

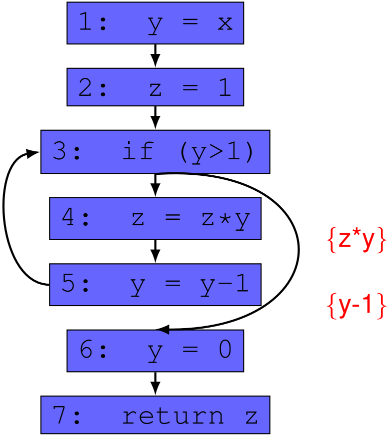
Constraint Systems



$$
\begin{aligned}
RD_{exit}(1) &\supseteq \{(y, 1)\} \\
RD_{entry}(3) &\supseteq RD_{exit}(2) \\
RD_{entry}(3) &\supseteq RD_{exit}(5)
\end{aligned}
$$

Example 2: Available Expressions

Which expressions have been computed and not changed since?

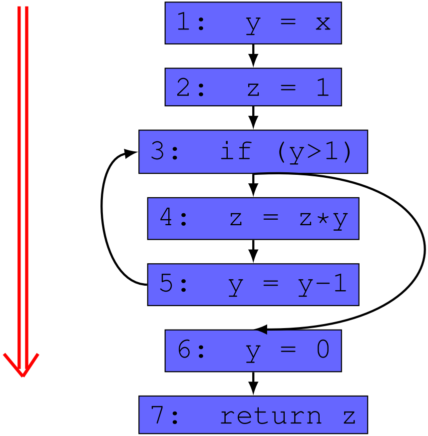


Available Expressions Problem Statement

An expression `v1 op v2` is available at statement *t* if it has been computed at statement *s* and all paths from *s* to *t* have no redefinitions of `v1` or `v2`.

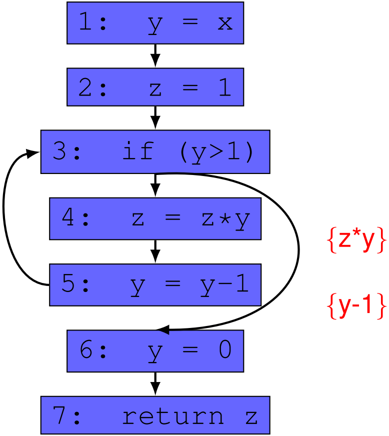
Available Expressions is a Forward Analysis

Expressions are available if they've already been computed.



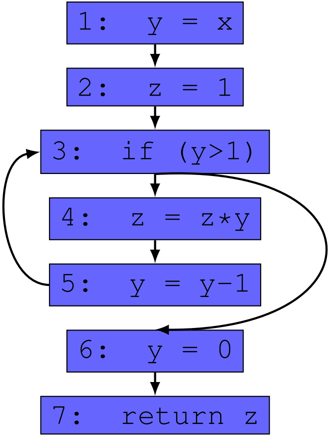
Available Expressions: Abstraction

Sets of expressions.



Available Expressions: Merge Operator

An expression must be available on all paths: intersection.



Available Expressions: Transfer Functions

For an all-paths analysis like available expressions:

$$\mathrm{out}(s) = \left(\bigcap_{i \in \mathrm{preds}(s)} \mathrm{out}(i) - \mathrm{kill}(s) \right) \cup \mathrm{gen}(s)$$

Available Expressions: Kill Sets

At an assignment statement,

$$s : v = \mathrm{RHS},$$

we kill all expressions containing v,

$$\text{e.g. } v + q, v * z, v.f$$

Available Expressions: Gen Sets

At a statement,

$$s : \cdots = \texttt{v1 op v2},$$

we generate the expression $\texttt{v1 op t2}$.

Available Expressions: Initial Values

- At entry points, no expressions are available; use $\emptyset$.
- At all other program points, assume all expressions available; use $\top$.