COSE312: Compilers

Lecture 15 — Data-Flow Analysis (1)

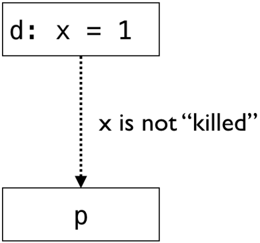
Hakjoo Oh
2015 Fall

Data-Flow Anlaysis

A program analysis technique that derives information about the flow of data along program execution paths.

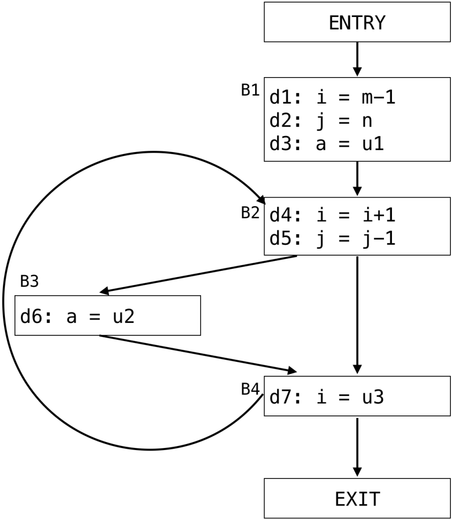
Reaching Definitions Analysis

- A definition d *reaches* a point p if there is a path from the definition point to p such that d is not "killed" along that path.



- For each program point, RDA finds definitions that *can* reach the program point along some execution paths.

Reaching Definitions Example



IN(B1) = {}

OUT(B1) = {}

IN(B2) = {}

OUT(B2) = {}

IN(B3) = {}

OUT(B3) = {}

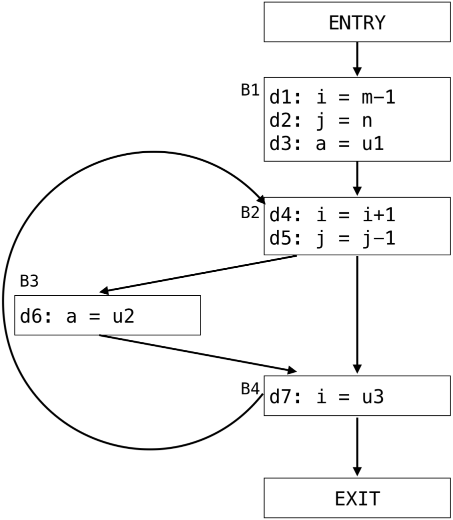
IN(B4) = {}

OUT(B4) = {}

IN(EXIT) = {}

OUT(EXIT) = {}

Reaching Definitions Example



IN(B1) = {}

OUT(B1) = {d1,d2,d3}

IN(B2) = {d1,d2,d3,d5,d6,d7}

OUT(B2) = {d3,d4,d5,d6}

IN(B3) = {d3,d4,d5,d6}

OUT(B3) = {d4,d5,d6}

IN(B4) = {d3,d4,d5,d6}

OUT(B4) = {d3,d5,d6,d7}

IN(EXIT) = {d3,d5,d6,d7}

OUT(EXIT) = {d3,d5,d6,d7}

The Analysis is Conservative

- Exact reaching definitions information cannot be obtained at compile time. It can be obtained only at runtime.
- ex) Deciding whether each path can be taken is undecidable:

```
a = rand(); b = rand(); c = rand();
if (a^10 + b^10 != c^10) {     // always true
  // (1)
} else {
  // (2)
}
```

- RDA computes an over-approximation of the reaching definitions that can be obtained at runtime.

Reaching Definitions Analysis

The goal is to compute

$$\begin{array}{rcl} \textbf{in} & : & Block \rightarrow 2^{Definitions} \\ \textbf{out} & : & Block \rightarrow 2^{Definitions} \end{array}$$

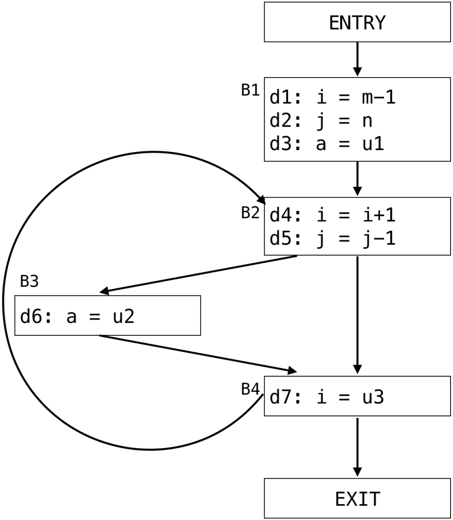
1. Compute gen/kill sets.
2. Derive transfer functions for each block in terms of gen/kill sets.
3. Derive the set of data-flow equations.
4. Solve the equation by the iterative fixed point algorithm.

1. Compute Gen/Kill Sets

$$\begin{array}{rcl} \textbf{gen} & : & Block \rightarrow 2^{Definitions} \\ \textbf{kill} & : & Block \rightarrow 2^{Definitions} \end{array}$$

- **gen**(B): the set of definitions "generated" at block B
- **kill**(B): the set of definitions "killed" at block B

Example



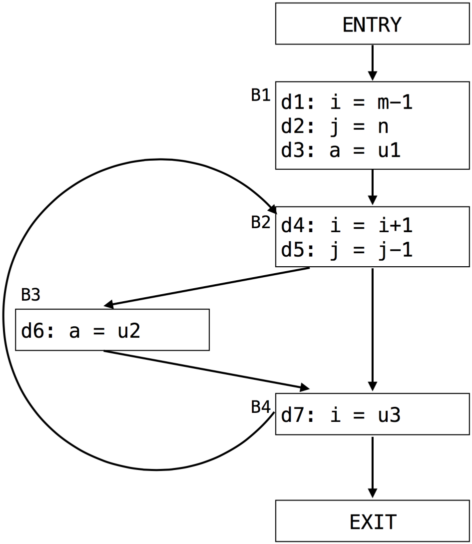
gen(B1) =
kill(B1) =

gen(B2) =
kill(B2) =

gen(B3) =
kill(B3) =

gen(B4) =
kill(B4) =

Example



gen(B1) = {d1,d2,d3}
kill(B1) = {d4,d5,d6,d7}

gen(B2) = {d4,d5}
kill(B2) = {d1,d2,d7}

gen(B3) = {d6}
kill(B3) = {d3}

gen(B4) = {d7}
kill(B4) = {d1,d4}

Exercise

Compute the **gen** and **kill** sets for the basic block B:

```
d1: a = 3
d2: a = 4
```

- $\textbf{gen}(B) =$
- $\textbf{kill}(B) =$

In general, when we have k definitions in a block B:

```
d1; d2; ... d_k
```

- $\textbf{gen}(B) =$
  $\textbf{gen}(d_k) \cup (\textbf{gen}(d_{k-1}) - \textbf{kill}(d_k)) \cup (\textbf{gen}(d_{k-2} - \textbf{kill}(d_{k-1}) - \textbf{kill}(d_k)) \cup \cdots \cup (\textbf{gen}(d_1) - \textbf{kill}(d_2) - \textbf{kill}(d_3) - \cdots - \textbf{kill}(d_k))$
- $\textbf{kill}(B) = \textbf{kill}(d_1) \cup \textbf{kill}(d_2) \cup \cdots \cup \textbf{kill}(d_k)$

2. Transfer Functions

$$f_B : 2^{Definitions} \to 2^{Definitions}$$

- The transfer function for a block B encodes the semantics of the block B, i.e., how the block transfers the input to the output.

$B2$

```
d4: i = i+1     {d1,d2,d3,d5,d6,d7}
d5: j = j-1     {d3,d4,d5,d6}
```

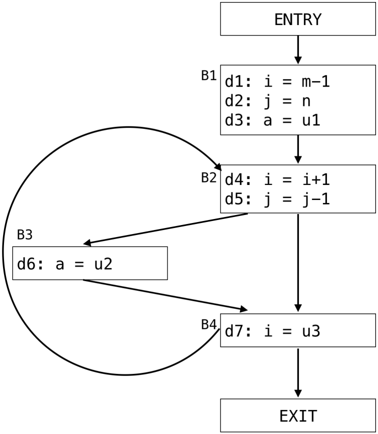
- The semantics of B is to add $\mathbf{gen}(B)$ and remove $\mathbf{kill}(B)$:

$$f_B(X) = \mathbf{gen}(X) \cup (X - \mathbf{kill}(X))$$

$B2$

```
d4: i = i+1     gen(B2) = {d4,d5}
d5: j = j-1     kill(B2) = {d1,d2,d7}
```

3. Derive Data-Flow Equations



$$
\begin{aligned}
\mathbf{in}(B_1) &= \emptyset \\
\mathbf{out}(B_1) &= f_{B_1}(\mathbf{in}(B_1))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{in}(B_2) &= \mathbf{out}(B_1) \cup \mathbf{out}(B_4) \\
\mathbf{out}(B_2) &= f_{B_2}(\mathbf{in}(B_2))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{in}(B_3) &= \mathbf{out}(B_2) \\
\mathbf{out}(B_3) &= f_{B_3}(\mathbf{in}(B_3))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{in}(B_4) &= \mathbf{out}(B_2) \cup \mathbf{out}(B_3) \\
\mathbf{out}(B_4) &= f_{B_4}(\mathbf{in}(B_4))
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{in}(B_i) &= \textstyle\bigcup_{P \hookrightarrow B_i} \mathbf{out}(P) \\
\mathbf{out}(B_i) &= f_B(\mathbf{in}(B_i)) \\
&= \mathbf{gen}(B_i) \cup (\mathbf{in}(B_i) - \mathbf{kill}(B_i))
\end{aligned}
$$

4. Solve the Equations

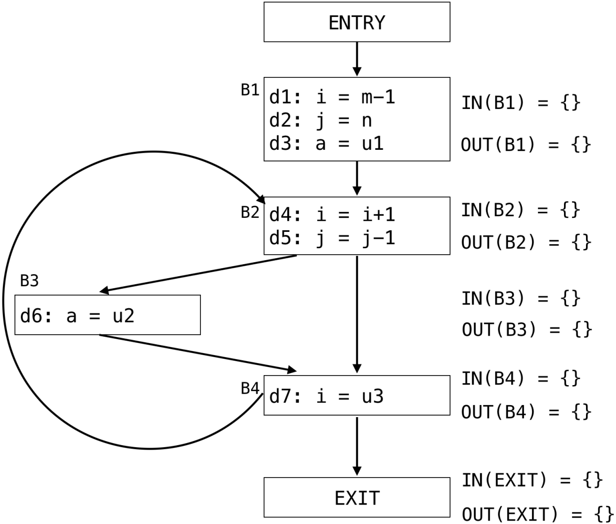
- The desired solution is the *least* **in** and **out** that satisfies the equations (why least?):

$$\begin{array}{rcl} \mathbf{in}(B_i) & = & \bigcup_{P \hookrightarrow B_i} \mathbf{out}(P) \\ \mathbf{out}(B_i) & = & \mathbf{gen}(B_i) \cup (\mathbf{in}(B_i) - \mathbf{kill}(B_i)) \end{array}$$

- The equations are solved by the iterative fixed point algorithm:

For all i, $\mathbf{in}(B_i) = \mathbf{out}(B_i) = \emptyset$
while (changes to any **in** and **out** occur) {
  For all i, update
    $\mathbf{in}(B_i) = \bigcup_{P \hookrightarrow B_i} \mathbf{out}(P)$
    $\mathbf{out}(B_i) = \mathbf{gen}(B_i) \cup (\mathbf{in}(B_i) - \mathbf{kill}(B_i))$
}

Example


ENTRY
B1
d1: i = m-1
d2: j = n
d3: a = u1
IN(B1) = {}
OUT(B1) = {}
B2
d4: i = i+1
d5: j = j-1
IN(B2) = {}
OUT(B2) = {}
B3
d6: a = u2
IN(B3) = {}
OUT(B3) = {}
B4
d7: i = u3
IN(B4) = {}
OUT(B4) = {}
EXIT
IN(EXIT) = {}
OUT(EXIT) = {}

cf) Reaching Definitions Analysis in Fixed Point Form

The reaching definitions information is defined as $\mathit{fix}\,F$, where F is defined as follows:

$$F(\mathbf{in}, \mathbf{out}) = (\lambda B.\ \bigcup_{P \hookrightarrow B} \mathbf{out}(P), \lambda B. f_B(\mathbf{in}(B)))$$

The least fixed point $\mathit{fix}\,F$ is by

$$\bigcup_{i \geq 0} F^i(\lambda B.\emptyset, \lambda B.\emptyset)$$

Summary

Every static analysis follows two steps:

1. Set up a set of *abstract semantic equations*.
   - about dynamics of program executions (e.g., how definitions flow)
2. Solve the equations using the iterative fixed point algorithm.
   - naive tabulation algorithm, worklist algorithm, etc