50.054 Liveness Analysis

ISTD, SUTD

Learning Outcomes

1. Define the liveness analysis problem
2. Apply lattice and fixed point algorithm to solve the liveness analysis problem

Recall

```
// SIMP1
x = input;
y = 0;
s = 0;
while (y < x) {
    y = y + 1;
    t = s;  // t is not used.
    s = s + y;
}
return s;
```

Liveness Analysis

- A variable x is *live* at program location v if it is used at program location u, and there exists a path $path(v, u)$ in the CFG.
- For instance, in SIMP1, y may be live bewteen line 2 and line 7. In other words, y must be dead at line 1, and line 9.

```
// SIMP1
x = input;
y = 0;              // y may be live
s = 0;              // y may be live
while (y < x) {     // y may be live
    y = y + 1;      // y may be live
    t = s;          // y may be live
    s = s + y;      // y may be live
}
return s;
```

Constructing Monotonic function for liveness analysis

1. Define the abstract domain, i.e. the abstract states.

Let V denote the set of variables in the input PA program. We define the abstract state as a set of variables that may be live.

2. Show that the abstract domain a complete lattice.

$(\mathcal{P}(V), \subseteq)$ is a powerset lattice. Since V is finite, it is a complete lattice.

3. Define the *join* function among abstract states. For liveness analysis

$$join(s_i) = \bigsqcup_{(i,j) \in CFG} s_j$$

In short we write

$$join(s_i) = \bigsqcup succ(s_i)$$

Constructing Monotonic function for liveness analysis

4. The monotonic functions can be defined by the following cases.

- case $l : ret,\ s_l = \{\}$
- case $l : t \leftarrow src,\ s_l = join(s_l) - \{t\} \cup var(src)$
- case $l : t \leftarrow src_1\ op\ src_2,\ s_l = join(s_l) - \{t\} \cup var(src_1) \cup var(src_2)$
- case $l : r \leftarrow src,\ s_l = join(s_l) \cup var(src)$
- case $l : r \leftarrow src_1\ op\ src_2,\ s_l = join(s_l) \cup var(src_1) \cup var(src_2)$
- case $l : ifn\ t\ goto\ l',\ s_l = join(s_l) \cup \{t\}$
- other cases: $s_l = join(s_l)$

$var(src)$ returns the list of variables used in the PA operand src.

Constructing Monotonic function for liveness analysis

```
// SIMP1
x = input;
y = 0;
s = 0;
while (y < x) {
    y = y + 1;
    t = s;
    s = s + y;
}
return s;
```

```
// PA1
1: x <- input
2: y <- 0
3: s <- 0
4: b <- y < x
5: ifn b goto 10
6: y <- y + 1
7: t <- s
8: s <- s + y
9: goto 4
10: rret <- s
11: ret
```

Constructing Monotonic function for liveness analysis

- case $l : ret$, $s_l = \{\}$
- case $l : t \leftarrow src$, $s_l = join(s_l) - \{t\} \cup var(src)$
- case $l : t \leftarrow src_1\ op\ src_2$, $s_l = join(s_l) - \{t\} \cup var(src_1) \cup var(src_2)$
- case $l : r \leftarrow src$, $s_l = join(s_l) \cup var(src)$
- case $l : r \leftarrow src_1\ op\ src_2$, $s_l = join(s_l) \cup var(src_1) \cup var(src_2)$
- case $l : ifn\ t\ goto\ l'$, $s_l = join(s_l) \cup \{t\}$
- other cases: $s_l = join(s_l)$

```
// PA1
1: x <- input // s1 = join(s1) - {x} U {input}
2: y <- 0     // s2 = join(s2) - {y}
3: s <- 0     // s3 = join(s3) - {s}
4: b <- y < x // s4 = join(s4) - {b} U {y,x}
5: ifn b goto 10 // s5 = join(s5) U {b}
6: y <- y + 1 // s6 = join(s6) - {y} U {y}
7: t <- s     // s7 = join(s7) - {t} U {s}
8: s <- s + y // s8 = join(s8) - {s} U {s,y}
9: goto 4     // s9 = join(s9)
10: rret <- s // s10 = join(s10) U {s}
11: ret       // s11 = {}
```

Constructing Monotonic function for liveness analysis

- case $l : ret$, $s_l = \{\}$
- case $l : t \leftarrow src$, $s_l = join(s_l) - \{t\} \cup var(src)$
- case $l : t \leftarrow src_1\ op\ src_2$,
  $s_l = join(s_l) - \{t\} \cup var(src_1) \cup var(src_2)$
- case $l : r \leftarrow src$, $s_l = join(s_l) \cup var(src)$
- case $l : r \leftarrow src_1\ op\ src_2$,
  $s_l = join(s_l) \cup var(src_1) \cup var(src_2)$
- case $l : ifn\ t\ goto\ l'$, $s_l = join(s_l) \cup \{t\}$
- other cases: $s_l = join(s_l)$

```
// PA1
1: x <- input // s1 = s2 - {x} U {input}
2: y <- 0     // s2 = s3 - {y}
3: s <- 0     // s3 = s4 - {s}
4: b <- y < x // s4 = s5 - {b} U {y,x}
5: ifn b goto 10 // s5 = s6 U s10 U {b}
6: y <- y + 1 // s6 = s7 - {y} U {y}
7: t <- s     // s7 = s8 - {t} U {s}
8: s <- s + y // s8 = s9 - {s} U {s,y}
9: goto 4     // s9 =  s4
10: rret <- s // s10 = s11 U {s}
11: ret       // s11 = {}
```

Constructing Monotonic function for liveness analysis

$$f_1(s_{11}, s_{10}, s_9, s_8, s_7, s_6, s_5, s_4, s_3, s_2, s_1) = \begin{pmatrix} \{\}, \\ \{s\}, \\ s_4, \\ (s_9 - \{s\}) \cup \{s, y\}, \\ (s_8 - \{t\}) \cup \{s\}, \\ (s_7 - \{y\}) \cup \{y\}, \\ s_6 \cup s_{10} \cup \{b\}, \\ (s_5 - \{b\}) \cup \{y, x\}, \\ s_4 - \{s\}, \\ s_3 - \{y\}, \\ (s_2 - \{x\}) \cup \{input\} \end{pmatrix}$$

$f_1 : \mathcal{P}(V)^{11} \to \mathcal{P}(V)^{11}$ is monotonic.

```
// PA1
1: x <- input // s1 = s2 - {x} U {input}
2: y <- 0     // s2 = s3 - {y}
3: s <- 0     // s3 = s4 - {s}
4: b <- y < x // s4 = s5 - {b} U {y,x}
5: ifn b goto 10 // s5 = s6 U s10 U {b}
6: y <- y + 1 // s6 = s7 - {y} U {y}
7: t <- s     // s7 = s8 - {t} U {s}
8: s <- s + y // s8 = s9 - {s} U {s,y}
9: goto 4     // s9 =  s4
10: rret <- s // s10 = s11 U {s}
11: ret       // s11 = {}
```

Applying Fixed Point Algorithm to liveness analysis

$$f_1(s_{11}, s_{10}, s_9, s_8, s_7, s_6, s_5, s_4, s_3, s_2, s_1) = \begin{pmatrix} \{\}, \\ \{s\}, \\ s_4, \\ (s_9 - \{s\}) \cup \{s, y\}, \\ (s_8 - \{t\}) \cup \{s\}, \\ (s_7 - \{y\}) \cup \{y\}, \\ s_6 \cup s_{10} \cup \{b\}, \\ (s_5 - \{b\}) \cup \{y, x\}, \\ s_4 - \{s\}, \\ s_3 - \{y\}, \\ (s_2 - \{x\}) \cup \{input\} \end{pmatrix}$$

$f_1 : \mathcal{P}(V)^{11} \to \mathcal{P}(V)^{11}$ is monotonic.

```
// PA1
1: x <- input // s1 = {input}
2: y <- 0     // s2 = {x}
3: s <- 0     // s3 = {y,x}
4: b <- y < x // s4 = {y,x,s}
5: ifn b goto 10 // s5 = {y,x,s,b}
6: y <- y + 1 // s6 = {y,x,s}
7: t <- s     // s7 = {y,x,s}
8: s <- s + y // s8 = {y,x,s}
9: goto 4     // s9 = {y,x,s}
10: rret <- s // s10 = {s}
11: ret       // s11 = {}
```

Forward vs Backward Analysis

- Forward Analysis - current state depends on predecessor (as per CFG) states, e.g. Sign Analysis
- Backward Analysis - current state depends on successor (as per CFG) states, e.g. Liveness Analysis

May vs Must Analysis

For analyses using powerset lattice,

- it is a *may* analysis if it gives an over-approximation, using $\cup$
  - liveness analysis - variables may be live.
- it is a *must* analysis if it gives an under-approximiation, using $\cap$.
  - deadness analysis - variable must be dead.

Deadness analysis is a dual of liveness analysis

1. Define the abstract domain, i.e. the abstract states.

Let V denote the set of variables in the input PA program. We define the abstract state as a set of variables that must be dead.

2. Show that the abstract domain a complete lattice.

$(\mathcal{P}(V), \subseteq)$ is a powerset lattice. Since V is finite, it is a complete lattice.

3. Define the *join* function among abstract states.

$$join(s_i) = \sqcap_{(i,j) \in CFG} s_j$$

Deadness analysis is a dual of liveness analysis

4. The monotonic functions can be defined by the following cases.

- case $l : ret,\ s_l = V$
- case $l : t \leftarrow src,\ s_l = join(s_l) - var(src) \cup \{t\}$
- case $l : t \leftarrow src_1\ op\ src_2,\ s_l = join(s_l) - (var(src_1) \cup var(src_2)) \cup \{t\}$
- case $l : r \leftarrow src,\ s_l = join(s_l) - var(src)$
- case $l : r \leftarrow src_1\ op\ src_2,\ s_l = join(s_l) - (var(src_1) \cup var(src_2))$
- case $l : ifn\ t\ goto\ l',\ s_l = join(s_l) - \{t\}$
- other cases: $s_l = join(s_l)$