# ECE 467 Midterm 2

University of Toronto

2022 November 18

1. (4 points) Consider the following grammar.

```
S -> A
S -> B
A -> C A
A -> a
B -> D B
B -> b
C -> d
D -> d
```

| Nonterminal | First |
|---:|---:|
| S | a, b, d |
| A | a, d |
| B | b, d |
| C | d |
| D | d |

(a) (2 points) Compute the LR(1) start state.

- -0.5 for each incorrect or missing item (no negative marks).

```
S' -> . S, $
S -> . A, $
S -> . B, $
A -> . C A, $
A -> . a, $
B -> . D B, $
B -> . b, $
C -> . d, a/d
D -> . d, b/d
```

(b) (2 points) Compute all the possible next states for the following state only (compute the GOTO for each symbol, and take the closure of each resulting kernel).

```
A -> C . A, $
A -> . C A, $
A -> . a, $
C -> . d, a/d
```

- -0.5 for each incorrect or missing state (no negative marks).

```
GOTO(_, A) = { [A -> C A ., $] }
// CLOSURE
{ [A -> C A ., $] }
```

```
GOTO(_, C) = { [A -> C . A, $] }
// CLOSURE
{ [A -> C . A, $], [A -> . C A, $], [A -> . a, $], [C -> . d, a/d] }

GOTO(_, a) = { [A -> a ., $] }
// CLOSURE
{ [A -> a ., $] }

GOTO(_, d) = { [C -> d ., a/d] }
// CLOSURE
{ [C -> d ., a/d] }
```

2. (4 points) Consider the following grammar, and the following LR(1) states (these are just a subset of all the LR(1) states for this grammar).

```
// Grammar
1. S -> E
2. E -> E - T
3. E -> T
4. T -> n
5. T -> l E r

// States
1. {[S -> E ., $], [E -> E . - T, $/-]}
2. {[E -> T ., $/-]}
3. {[T -> n ., $/-]}
4. {[T -> l E . r, $/-], [E -> E . - T, r/-]}
5. {[E -> T ., r/-]}
6. {[T -> l E r ., $/-]}
7. {[T -> l E . r, r/-], [E -> E . - T, r/-]}
```

(a) (2 points) Draw the LR(1) parsing table for the above states, filling in just the reduces.

- -0.3 for each incorrect or missing row (no negative marks).

| State | r | - | $ |
|---|---|---|---|
| 1 | | | r1 |
| 2 | | r3 | r3 |
| 3 | | r4 | r4 |
| 4 | | | |
| 5 | r3 | r3 | |
| 6 | | r5 | r5 |
| 7 | | | |

(b) (2 points) Draw the merged LALR(1) states from those LR(1) states.

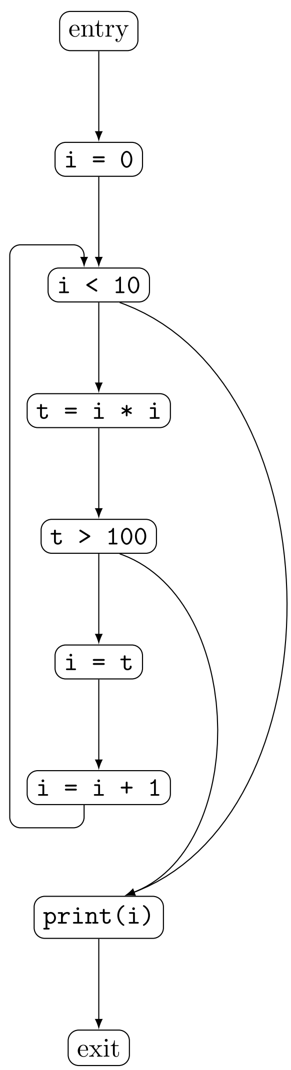
- -0.5 for each incorrect or missing state (no negative marks).

```
1: {[S -> E ., $], [E -> E . - T, $/-]}
2/5: {[E -> T ., r/$/-]}
3: {[T -> n ., $/-]}
4/7: {[T -> l E . r, r/$/-], [E -> E . - T, r/-]}
6: {[T -> l E r ., $/-]}
```

3. (2 points) Draw a control-flow graph for the following code.

```
int i;
for (i = 0; i < 10; i = i + 1) {
    t = i * i;
    if (t > 100) {
        break;
    }
    i = t;
}
print(i);
```

- -0.5 for each mistake.



4. (4 points) For busy expressions analysis:

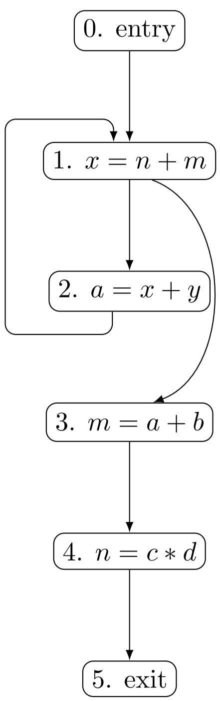
- A value in the domain is a set of expressions that appear in the program.
- It is a backwards analysis.
- $f_s(x) = \text{gen}(s) \cup (x \setminus \text{kill}(s))$.
- gen($s$) is the set containing the expression of $s$.
- kill($s$) is the set of all expressions in the program that have an operand that is assigned to by $s$.
- Meet is set intersection.
- Initialize IN[exit] to be empty. Initialize IN of every other node to be the set of all expressions.

The iterative algorithm for backwards analysis is as follows.

```
// initialization
while changed {
    for each node s in the CFG {
        OUT[s] = meet IN[s'] for all successors s';
        IN[s] = f_s(OUT[s]);
    }
}
```

Compute 1 iteration of busy expressions analysis, going in *backwards order* (5, 4, 3, 2, 1, 0). *Reminder*: you don't need to do anything for OUT[exit] or IN[entry].

- -0.5 for each incorrect IN or OUT.




| Node | Gen | Kill | $\text{IN}_0$ | $\text{OUT}_1$ | $\text{IN}_1$ |
|---|---|---|---|---|---|
| 5 | n/a | n/a | $\emptyset$ | n/a | $\emptyset$ |
| 4 | $\{c * d\}$ | $\{n + m\}$ | $\{n + m, x + y, a + b, c * d\}$ | $\emptyset$ | $\{c * d\}$ |
| 3 | $\{a + b\}$ | $\{n + m\}$ | $\{n + m, x + y, a + b, c * d\}$ | $\{c * d\}$ | $\{c * d, a + b\}$ |
| 2 | $\{x + y\}$ | $\{a + b\}$ | $\{n + m, x + y, a + b, c * d\}$ | $\{n + m, x + y, a + b, c * d\}$ | $\{n + m, x + y, c * d\}$ |
| 1 | $\{n + m\}$ | $\{x + y\}$ | $\{n + m, x + y, a + b, c * d\}$ | $\{c * d\}$ | $\{c * d, n + m\}$ |
| 0 | n/a | n/a | n/a | $\{c * d, n + m\}$ | n/a |

5. (2 points) A semilattice consists of the following.

   - A domain (a set of values) $V$.
   - A binary meet $\wedge$ operator.
   - A distinguished "top" value $\top$.

   The meet operator must satisfy the following relations, for all $x, y, z \in V$.

   1. $x \wedge x = x$.
   2. $x \wedge y = y \wedge x$.
   3. $(x \wedge y) \wedge z = x \wedge (y \wedge z)$.

   Additionally, for all values $x \in V$, we must have that $\top \wedge x = x$.

   Suppose $\top' \in V$ also satisfies $\top' \wedge x = x$ for all $x \in V$. Can $\top'$ be different from $\top$ (does the property for "top" in a lattice uniquely define an element)? Prove or disprove.

   - 1 point if you leave this blank; otherwise
   - -1 for incorrect reasoning, -0.5 for unclear or incomplete reasoning.

   Top $\top$ satisfies $\top \wedge x = x$ for all $x$. In particular, setting $x = \top'$, we get $\top \wedge \top' = \top'$.

   Since $\top'$ also satisfies $\top' \wedge x = x$ for all $x$, setting $x = \top$, we get $\top' \wedge \top = \top$.

   By commutativity, $\top \wedge \top' = \top' \wedge \top$. So:

$$\top' = \top \wedge \top' = \top' \wedge \top = \top.$$

   Top is unique.

6. (4 points) *In general*, for any set $S$, a partial order $\leq$ is a binary relation on $S$ such that for all $x, y, z \in S$, the following hold.

   1. $x \leq x$.
   2. $x \leq y$ and $y \leq x$ implies $x = y$.
   3. $x \leq y$ and $y \leq z$ implies $x \leq z$.

   *In general*, given a partial order $\leq$ on a set $S$, and a function $f : S \to S$, we say $f$ is *monotonic* if and only if:

$$x \leq y \text{ implies } f(x) \leq f(y).$$

   *Note*: in the case of data-flow analysis, the transfer indeed maps values from the domain of a lattice $V$ back to $V$ (it remains to be shown for each analysis that the transfer functions do actually satisfy this property).

   *For a (semi)lattice*, we can define a partial order on its domain $V$ as follows:

$$x \leq y \text{ if and only if } x \wedge y = x.$$

   Suppose we have a function $f$ and a lattice with domain $V$ that satisfies the following (for all $x, y \in V$).

$$f(x \wedge y) \leq f(x) \wedge f(y).$$

   Let $a, b \in V$, such that $a \leq b$. Prove that $f(a) \leq f(b)$.

   For the following parts worth 1 point each:

   - 0.3 points if you leave it blank; otherwise:
   - 0 for incorrect, incomplete, or unclear reasoning.

   (a) (1 point) Rewrite $a \leq b$ in terms of the meet operator.

$$a \wedge b = a.$$

   (b) (1 point) Rewrite $f(x \wedge y) \leq f(x) \wedge f(y)$ in terms of the meet operator.

$$f(x \wedge y) \wedge (f(x) \wedge f(y)) = f(x \wedge y).$$

   (c) (1 point) Rewrite $f(a) \leq f(b)$ in terms of the meet operator.

$$f(a) \wedge f(b) = f(a).$$

   (d) (1 point) Complete the rest of the proof.

   The statements in parts (a) and (b) are true by assumption/definitions. The statement in part (c) is (equivalent to) what we want to prove.

   - Set $x = a$ and $y = b$.
   - By (b), $f(a \wedge b) \wedge f(a) \wedge f(b) = f(a \wedge b)$.
   - By (a), $f(a) \wedge f(a) \wedge f(b) = f(a)$.
   - By idempotency of meet, $f(a) \wedge f(b) = f(a)$.
   - By (c), this is equivalent to $f(a) \leq f(b)$, as desired.

7. (3 points) A *partition* $P$ of a set $S$ is a set of subsets of $S$, such that every element $x \in S$ appears in one and only one subset of $S$ in the partition $P$.

   For example, given the set $\{ a, b, c, d, e \}$, the following are possible (but not all) partitions:

   - $\{ \{ a, b \}, \{ c, d \}, \{ e \} \}$.
   - $\{ \{ a \}, \{ b \}, \{ c \}, \{ d \}, \{ e \} \}$.
   - $\{ \{ a, b, c, d, e \} \}$.

   (Recall a partition is a set of (sub)sets.) Given two partitions $P_1$ and $P_2$ of a set $S$, we first define a partial order:

   $$P_1 \leq P_2 \text{ iff every element of } P_1 \text{ is a subset of an element of } P_2.$$

   Given two elements $x, y \in S$ and a partition $P$ of $S$, we say $x \cong_P y$ ($x$ and $y$ are equivalent with respect to the partition $P$) iff $x$ and $y$ are in a single set in the partition $P$ (an element of $P$ is a set).

   (a) (1 point) Write down any lower bound for the following two partitions based on the partial order defined.

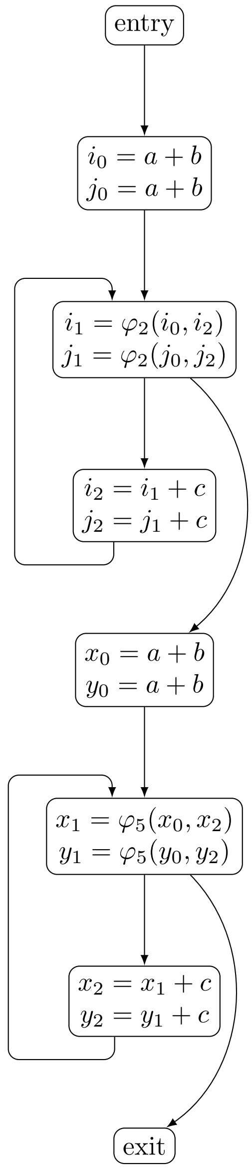
   - $P_0 = \{ \{ i_0, j_0, i_1, j_1 \} \}$,
   - $P_1 = \{ \{ i_0, j_0 \}, \{ i_1, j_1 \} \}$.
   - No penalty for error.

   $P_1$ is a lower bound for $P_0$ ($P_1 \leq P_0$) and $P_1$ ($P_1 \leq P_1$).

(b) (2 points) Consider a new data-flow analysis for a program in SSA form. Recall that each variable/name in SSA form is unique; a name is defined by a single expression. Let $S$ be the set of all names (and their corresponding expressions) in the program.

- A value in the domain is a partition of $S$.
- The analysis is forwards.
- The meet of two values $P_1, P_2$ in the domain is the *greatest lower bound* of $P_1$ and $P_2$ (you can imagine computing this by brute force).
- Assume every name/defining expression $s$ takes the form of $s = \text{op}(u, v)$ where op is a deterministic, side-effect-free function in the program. The transfer function $f_s(P)$ is as follows.
  - $s$ is a name/expression in the program. So it is in $S$. So it is in exactly one set in the partition $P$. Designate that set containing $s$ as $X$.
  - The result $f_s(P)$ will be a partition containing all elements of $P$ except $X$, and instead of $X$ we will have $X_1$ and $X_2$, where we split $X$ into the two sets $X_1$ and $X_2$.
  - We split $X$ as follows: start by placing $s$ in $X_1$. Then for all other names/expressions $s' = \text{op}'(u', v') \in X$:
  - We place $s'$ in $X_1$ *only if* $s$ and $s'$ have the same operator, $u \cong_P u'$, and $v \cong_P v'$ (all three hold). We place $s'$ in $X_2$ otherwise. (We loop over $X$ once and that's it.)
  - For example, if $s = a - b$ and $P$ is the set containing $\{ s = a - b, y = c - d, z = e - f \}$, $\{ a = ..., c = ..., f = ... \}$, and $\{ b = ..., d = ..., e = ... \}$:
  - Then $f_s(P)$ contains $\{ a = ..., c = ..., f = ... \}$ and $\{ b = ..., d = ..., e = ... \}$ verbatim, and $\{ s = a - b, y = c - d, z = e - f \}$ needs to be split.
  - We first put $s = a - b$ in $X_1$.
  - Consider $y = c - d$. It has the same operator. $c$ (the first operand of $y$) was in the same set as $a$ (the first operand of $s$) (referring back to $P$). $d$ was in the same set as $b$. So we put $y = c - d$ in $X_1$.
  - Consider $z = e - f$. It has the same operator. But $e$ ($z$'s first operand) is not in the same set as $a$ ($s$'s first operand). So we put $z = e - f$ in $X_2$.
  - Our result $f_s(P)$ is the set containing $\{ s = a - b, y = c - d \}$, $\{ z = e - f \}$, $\{ a = ..., c = ..., f = ... \}$, and $\{ b = ..., d = ..., e = ... \}$.

Finally, consider the following program (suppose $a, b, c$ are some previously defined values).




Consider $+$ to be the usual integer operator/function, and note that $\varphi_2$ and $\varphi_5$ are different from each other (but the two $\varphi_2$ in the same basic block are the same operator, and similarly for $\varphi_5$ in the other basic block).

Break the basic blocks into individual nodes per instruction.

Initialize OUT for all nodes to be the partition $\{ \{ i_0, j_0, i_2, j_2, x_0, y_0, x_2, y_2, i_1, j_1, x_1, y_1 \}, \{ a \}, \{ b \}, \{ c \} \}$.

For forwards analysis:

```
// initialization
while changed {
    for each node s in the CFG {
        IN[s] = meet OUT[s'] for all predecessors s';
        OUT[s] = f_s(IN[s]);
    }
}
```

Carry out the data-flow analysis for the first 4 nodes (after splitting the basic blocks) (you should have 4 INs and 4 OUTs).

- 0.5 if you leave it blank, otherwise
- -0.3 points per error.

```
0. entry
1. i0 = a + b
2. j0 = a + b
3. i1 = phi2(i0, i2)
4. j1 = phi2(j0, j2)
```

Note that the transfer function only splits one element/set of the partition at a time.

- $\text{IN}_1[1] = \text{OUT}_0[0] = \{\{i_0, j_0, i_2, j_2, x_0, y_0, x_2, y_2, i_1, j_1, x_1, y_1\}, \{a\}, \{b\}, \{c\}\}$.
- $\text{OUT}_1[1] = \{\{i_0, j_0, x_0, y_0\}, \{i_1, j_1, x_1, y_1, i_2, j_2, x_2, j_2\}, \{a\}, \{b\}, \{c\}\}$.
- $\text{IN}_1[2] = \text{OUT}_1[1]$.
- $\text{OUT}_1[2] = \text{IN}_1[2]$ (no change).
- $\text{IN}_1[3] = \text{OUT}_1[2]$ (the initial value meet $\text{OUT}_1[2]$ is $\text{OUT}_1[2]$).
- $\text{OUT}_1[3] = \{\{i_0, j_0, x_0, y_0\}, \{i_1, j_1\}, \{x_1, y_1, i_2, j_2, x_2, j_2\}, \{a\}, \{b\}, \{c\}\}$.
- $\text{IN}_1[4] = \text{OUT}_1[3]$.
- $\text{OUT}_1[4] = \text{IN}_1[4]$ (no change).

(c) (1 bonus points) What does this data-flow analysis compute?

- No penalty for error.

Names/expressions in the same set at the end of the iterative algorithm must have equivalent values.

(d) (1 bonus points) Why must the $\varphi$ in different basic blocks be considered different operators/functions ($\varphi_2$ and $\varphi_5$)? *Hint*: think about how the algorithm could produce incorrect results.

- No penalty for error.

The "action" of a phi function depends on the control-flow. The control-flow is different in different basic blocks. So phi functions in different basic blocks have different semantics.

If $\varphi_2$ and $\varphi_5$ are treated as the same operator, then the algorithm would think that $i_1 = j_1 = x_1 = y_1$, even though the loops they are in are different and could execute a different number of times.