CS201: Sample Solutions

1 Control flow analysis

1. Dominator sets:
   \[ D(1) = \{1\} \]
   \[ D(2) = \{1,2\} \]
   \[ D(3) = \{1,2,3\} \]
   \[ D(4) = \{1,2,4\} \]
   \[ D(5) = \{1,2,3,5\} \]
   \[ D(6) = \{1,2,3,5,6\} \]
   \[ D(7) = \{1,2,3,5,7\} \]
   \[ D(8) = \{1,2,3,5,6,8\} \]

Dominator tree:
2. There are three loops:

(a) For back edge 3 → 2, loop is {2,3};
(b) For back edge 8 → 5, loop is {5,6,7,8};
(c) For back edge 8 → 2, loop is {2,3,5,6,7,8}.

Therefore, the first two loops are nested in the third loop.

3. This control flow graph is not reducible. Because after we have removed all back edges (i.e., 3 → 2, 8 → 5 and 8 → 2), there is still a cycle {6,7} in the graph.

2 Depth first numbering

1. Depth first search: 1 → 2 → 3 → 8 → 2 → 8 → 3 → 2 → 4 → 5 → 7 → 2 → 4 → 6 → 4 → 2 → 1

   Depth first numbering: 124657389.


   Loops: {2,3,4,5,6,7,8} and {4,5,6,7}. 
Sample Problems: Data Flow Framework

1. Reachable uses – for each definition identify the set of uses reachable by the definition (used for computing def-use chains);

   - Data flow equations.
     This problem is similar to live variables. A use is reachable if there is a X-clear path from the definition to it.

     \[
     \begin{align*}
     \text{IN}[B] & \text{ uses live at B's entry.} \\
     \text{OUT}[B] & \text{ uses live at B's exit.} \\
     \{ & \text{OUT}[B] = \bigcup_{s \in \text{succ}(B)} \text{IN}[s] \\
     \text{IN}[B] & = \text{GEN}[B] \cup (\text{OUT}[B] - \text{KILL}[B])
     \end{align*}
     \]

     \text{GEN}[B]: uses in B prior to any definition of the same variable in B.
     \text{KILL}[B]: uses whose variables are redefined in B.

   - Lattice value. We can use \text{USES} to denote the set of all uses. So the lattice value \( L = \mathcal{P}(\text{USES}) \).
   - Meet operator \( \wedge: \cup \)
   - Top element: \( \emptyset \); bottom elements: \text{USES}.
   - Partial order relation: \( \supseteq \)
   - Pictorial representation of the partial order. \text{USES} = \{u_1, u_2, u_3\}.

```
          _______      _______      _______
         |      |      |      |
        /      /      /      /
       |      |      |      |
      (u_1)  (u_2)  (u_3)  (u_4)
        |      |      |      |
       /      /      /      /
      (u_1, u_2) (u_1, u_3) (u_2, u_3) (u_4, u_3)
        |      |      |      |
       /      /      /      /
      (u_1, u_2, u_3) (u_1, u_2, u_3) (u_2, u_3, u_4) (u_4, u_3, u_4)
        |      |      |      |
       /      /      /      /
     (u_1, u_2, u_3, u_4) (u_1, u_2, u_3, u_4) (u_2, u_3, u_4, u_5) (u_4, u_3, u_4, u_5)
```

   - Transfer functions.
     \[
     f_n(x) = \text{GEN}[n] \cup (x - \text{KILL}[n])
     \]

     \text{GEN}[n] = \{ u \in \text{USES}| n \text{ contains use } u \text{ whose variable is not redefined later in } n \}
     \text{KILL}[n] = \{ u \in \text{USES}| n \text{ defines the variable used by } u \}

2. Reaching uses – given a definition of variable \( x \), identify the set of uses of \( x \) that are encountered prior to reaching the definition and there is no other definitions of \( x \) that intervene the use and the definition (used for computing antidependences);
- Data flow equations.
  \[
  \begin{align*}
  \text{IN}[B] & = \bigcup_{s \in \text{pred}(B)} \text{OUT}[s] \\
  \text{OUT}[B] & = \text{GEN}[B] \cup (\text{IN}[B] - \text{KILL}[B])
  \end{align*}
  \]
  
  - \text{GEN}[B]: uses within B that reach the end of B.
  - \text{KILL}[B]: uses of the variables that are redefined in B.

- Lattice value. We can use USES to denote the set of all uses. So the lattice value \( \mathcal{L} = \mathcal{P}(\text{USES}) \).
- Meet operator \( \wedge: \cup \)
- Top element: \( \emptyset \); bottom elements: USES.
- Partial order relation: \( \supseteq \)
- Pictorial representation of the partial order. USES={u1, u2, u3}.

\[\begin{array}{c}
\top \\
\downarrow \\
\downarrow \\
\downarrow \\
(u1) \\
\downarrow \\
\downarrow \\
\downarrow \\
(u1, u2) \\
\downarrow \\
\downarrow \\
\downarrow \\
(u1, u3) \\
\downarrow \\
\downarrow \\
\downarrow \\
(u2, u3) \\
\downarrow \\
\downarrow \\
\downarrow \\
\downarrow \\
\bot
\end{array}\]

- Transfer functions.
  \[f_n(x) = \text{GEN}[n] \cup (x - \text{KILL}[n])\]
  \[
  \begin{align*}
  \text{GEN}[n] &= \{u \in \text{USES}| n \text{ contains use } u \text{ which reach the end of } n\} \\
  \text{KILL}[n] &= \{u \in \text{USES}| n \text{ defines the variable used by } u\}
  \end{align*}
  \]

3. Classify the value of each program variable at each program point into one of the following categories: (a) the value is a **unique constant** - you must also identify this constant value; (b) the value is **one-of-many** constants – you are not actually compute the identities of these constants as part of your solution; and (c) the value is **not-a-constant**, that is, it is neither a unique constant nor a one-of-many constants.

**Single Variable.**

- Lattice for a single variable.
• Meet operator for a single variable (using M to denote one-of-many):

\[
\begin{align*}
\text{any} \land T &= \text{any} \\
\text{any} \land \bot &= \bot \\
c \land c &= c \\
c_1 \land c_2 &= M \quad (c_1 \neq c_2) \\
c \land M &= M \\
M \land M &= M
\end{align*}
\]

Now, we consider all variables.

• \text{VAR}- set of all variables
• \text{VAL}- set of all values (all constants, M, \top, \bot)

Lattice for all variables \((L, \land')\).

• Lattice value.

\[\mathcal{L} = \{ \beta \mid \text{total function } \beta : \text{VAR} \rightarrow \text{VAL} \}\]

• Meet operator: \(\land'\).
  Definition of \(\land'\):
  - \(\beta_T\) maps each variable to \(\top\)
  - \(\beta_{\bot}\) maps each variable to \(\bot\)
  - \((\beta_1 \land' \beta_2)(v) = \beta_1(v) \land \beta_2(v)\)

• Top element: \(\beta_T\); bottom elements: \(\beta_{\bot}\).

• Partial order relation: \(\leq\)
  Definition of \(\leq\): \(\beta_1 \leq \beta_2\) iff \(\forall v \in \text{VAR}, \beta_1(v) \leq \beta_2(v)\)

• Transfer functions.
  For statement \(S : A = B\) op \(C\):

\[
\begin{align*}
\beta(v) &= \quad \forall v \in \text{VAR} - \{A\} \\
\beta(A) &= \top \quad \text{if } \beta(B) = \bot \text{ or } \beta(C) = \bot \quad \text{(no change)} \\
\beta(A) &= \bot \quad \text{else if } \beta(B) = T \text{ or } \beta(C) = T \quad \text{(not a constant)} \\
\beta(A) &= c_B \text{ op } c_C \quad \text{else if } \beta(B) = c_B \text{ and } \beta(C) = c_C \quad \text{(constant)}
\end{align*}
\]
- Data flow equations.
  IN[B]: the mapping of variables at B's entry.
  OUT[B]: the mapping of variables at B's exit.
  \[
  \begin{aligned}
  \{\text{IN}[B] &= \Lambda_{s \in \text{Pred}(B)} \text{OUT}[s] \\
  \text{OUT}[B] &= f_B(\text{IN}[B])
  \end{aligned}
  \]
  where \( f \) is the transfer function of block B

4. Postdominator set of a node is the set of nodes that are encountered along all paths from the node to the end node of the control flow graph;

- Data flow equations.
  OUT[B]: postdominators at B's exit.
  IN[B]: postdominators at B's entry.
  \[
  \begin{aligned}
  \{\text{OUT}[B] &= \cap_{s \in \text{succ}(B)} \text{IN}[s] \\
  \text{IN}[B] &= \{B\} \cup \text{OUT}[B]
  \end{aligned}
  \]

- Lattice value. We can use \( N \) to denote the set of nodes. So the lattice value \( \mathcal{L} = \mathcal{P}(N) \).
- Meet operator \( \land: \cap \)
- Top element: \( N \); bottom elements: \( \emptyset \).
- Partial order relation: \( \subseteq \)
- Pictorial representation of the partial order. \( N = \{n1, n2, n3\} \).

- Transfer functions.
  \[
  f_n(x) = \{n\} \cup x
  \]
Sample Problems and Solutions - II

1. For the code segment below:
   a) Compute the dominance frontier for all nodes;
   b) Identify the placement points of phi nodes for each variable; and
   c) Provide the final SSA-form.

```
P = 1
Q = 2

If()
P = P + Q
K = 1
K = 2
Q = Q + 1

K = P + 1

If()
P = K + P
Q = K + Q

If()
```
(b) Variable | Node where assigned value | Node where $\emptyset$ should be placed
--- | --- | ---
P | 1, 3, 7 | 2, 5, 6
Q | 1, 4, 7 | 2, 5, 6
K | 3, 4, 5 | 2, 5
(c)

\[ P_1 = 1 \]
\[ Q_1 = 3 \]

\[ P_2 = \phi (P_1, P_5) \]
\[ Q_2 = \phi (Q_1, Q_5) \]
\[ k \text{ could have a } \phi \text{ rule here, but it doesn't really matter} \]

\[ P_3 = P_2 + Q_2 \]
\[ k_1 = 1 \]

\[ k_2 = 2 \]
\[ Q_3 = Q_2 + 1 \]

\[ P_4 = \phi (P_3, P_2) \]
\[ Q_4 = \phi (Q_2, Q_3) \]
\[ k_3 = \phi (k_1, k_2) \]
\[ k_4 = P_4 + 1 \]

\[ P_5 = \phi (P_4, P_6) \]
\[ Q_5 = \phi (Q_4, Q_6) \]
\[ \text{LP}(\cdot) \]

\[ P_6 = k_4 + P_5 \]
\[ Q_6 = k_4 + Q_5 \]

\[ \text{LP}(\cdot) \]
Q.1. We would like to construct a constant propagation algorithm that identifies if the value of each variable at each program point is one of the following constants: -1, 0, +1. Which of the two lattices will you use in solving this problem? Provide a clear justification for your choice. Hint: Note that in the second lattice all constants other than -1, 0, +1 are treated as not-a-constant.

Q.2. Consider the restricted class of programs in which any expression on the right hand side of an assignment statement involves no more than one variable, i.e. other operands if present are constants. The data flow analysis for the constant propagation and folding problem for general programs is known to be non-distributive. If this also the case for the above restricted class of programs? Explain.
\[ \text{def} \quad \wedge: \quad \text{def} \wedge \text{def} = \text{def} \]
\[ \text{undef} \wedge \text{def} = \text{undef} \]
\[ \text{undef} \wedge \text{undef} = \text{undef} \]

\[ f_n(u): \]

\[ n: \quad \{ w = - \} \quad w \neq u \]

\[ f_n(u) = \text{id} \]

Initialisation:

\[ \forall u, \quad \text{IN}[s, u] = \text{undef} \quad \text{where} \ s \ \text{is the start node} \]

\[ \text{IN}[\text{def}, u] = \text{def} \]

Equations:

\[ \text{IN}[n, v] = \bigwedge_{p \in \text{pred}(n)} f_p(u)(\text{IN}[p, v]) \]

Definition:

\[ \forall \lambda, \ \text{if} \ n \ \text{uses} \ v \ \text{and} \ \text{IN}[n, v] = \text{undef} \]

then this use of \ v \ in \ n \ is \ a \ potentially \ undefined \ variable. \]

Q3. Develop an algorithm for identifying potential uses of uninitialized variables, i.e., uses of variables that are not always preceded by their definition.
Sample Problem

Algorithm:

1. Print $V+W$.
2. While ($B > 0$):
   4. Else $V = Z - 1$.
5. If ($Z > 0$) then $V = Z + 1$.
7. Do
8. Else $Z = B - A$.
9. If ($A = B$) then $Z = B + A$.
10. $A = 1$; $W = 1$; $B = Z$.

Time complexity: $O(n)$

(a) Using SSA
(b) Using GISA
Algorithm: and
Flow based
(c) Using data
Folding
Transformation and
Constant
Conditional
Perform
Code segment
For the given
Sample Problem

```
endwhile
X = A + 1; Y = B + 1;
A = A + 1; B = B + 1;
else V = A + Z
endif
W = A + 1; Z = Z + 1

if (...) then
  W = Z; R = Z + 1;
  Z = 1; P = Z + 1;
  while (...) do
    X = A + B; Y = B + A;
  endwhile
input A; input B;
```

```
W2 = v1 = x2
P1 = R1 = x1

\[ \begin{align*}
    A_1 &= B_2 + 1 \\
    x_2 &= A_2 + 1 \\
    B_2 &= B_3 + 1 \\
    A_2 &= A_3 + 1 \\
    W_3 &= (w_1 w_2)
\end{align*} \]

end

1 + z + 1 + 1 = 1
\( \text{else} \)
\( A_2 = z + 1 \)
\( W_2 = A_3 + 1 \)
\( ( - ) \)
\( \text{then} \)
\( R_1 = z_1 + 1 \\
    W_1 = z_2 \\
    p_1 = z_1 + 1 \\
    \varepsilon_1 = z_2 + 1 \)
\( \frac{7_3}{\phi_1 y_2} \)
\( x_3 = \frac{\phi_2}{x_3 x_2} \)
\( B_3 = \phi_2 \frac{B_1}{B_2} \)
\( A_3 = \phi_2 \frac{A_1 A_2}{\phi} \)

while (C)
\( \lambda_1 = B_1 + A \)
\( \frac{\lambda_1 x_1}{A_1 + B} \)
\( \lambda_1 = A + B \)
\( \lambda_1 = A + B \)
4.4. Code Optimization

Develop an algorithm that eliminates those fully redundant evaluations of an expression \( \text{expr} \), that can be eliminated without introducing new temporary variables in the program. The example below illustrates the optimization. Note that \( D = P + Q \) is replaced by \( D = Z \) eliminating redundancy without introducing a temporary variable; however, redundancy due to \( X + Y \) cannot be eliminated.
\[
\begin{align*}
\text{undef} & \quad \text{(variables)} \\
\text{not-available} & \quad (\text{not available in any variable}) \\
\end{align*}
\]

\[
\begin{align*}
\land & \quad \text{any } \land \text{undef } = \text{any} \\
\land & \quad v \land v = v \\
\land & \quad v \land \text{not-available } = \text{not-available} \\
\text{any } \land \text{not-available } = \text{not-available} \\
\end{align*}
\]

\[
\begin{align*}
f(n, \text{exp}) : & \quad (\text{generates}) \\
\Rightarrow & \quad \exists \text{kill exp which uses variable } v \\
& \quad f(n, \text{exp}) = \text{not-available} \\
\end{align*}
\]

\[
\begin{align*}
f(n, \text{exp}) : & \quad (\text{kills}) \\
\Rightarrow & \quad \forall \text{kill exp which uses variable } v \\
& \quad f(n, \text{exp}) = \text{identity} \\
\end{align*}
\]

\[
\begin{align*}
\text{Intraschema:} \\
\forall \text{exp}, \quad \text{IN}[s, \text{exp}] = \text{not-available} \quad \text{where } s \text{ is the start node} \\
\text{IN}[n \neq s, \text{exp}] = \text{undef} .
\end{align*}
\]

\[
\begin{align*}
\text{Equations:} \\
\text{IN}[n, \text{exp}] = \land_{p \in \text{prod}(n)} f(p, \text{exp}) (\text{IN}[p, \text{exp}])
\end{align*}
\]

\[
\begin{align*}
\text{Transformation:} \\
\text{for each assignment } n : u = \text{exp} \\
\text{if } \text{IN}[n, \text{exp}] = w \text{ then} \\
\text{replace } u = \text{exp} \text{ by } u = w.
\end{align*}
\]