Exercises

Program Analysis

Sheet 3

Exercise 1 Consider the following while program:

\[
\begin{align*}
&y := 2^1; \\
&\text{if } [z > 1]^2 \\
&\text{then } [x := 1]^3 \\
&\text{else } [x := -1]^4; \\
&[y := x \times x]^5;
\end{align*}
\]

Perform a Constant Propagation Analysis CP for this program.

Solution We represent the state at a label as

\[
\hat{\sigma} = [x \mapsto \sigma(x), y \mapsto \sigma(y), z \mapsto \sigma(z)] = \begin{array}{ccc}
  x & y & z \\
  \sigma(x) & \sigma(y) & \sigma(z)
\end{array}
\]

The transfer functions are given as

\[
\begin{align*}
\phi_{\text{CP}1} & : \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} = \begin{pmatrix} x & y & z \\ \sigma(x) & 2 & \sigma(z) \end{pmatrix} \\
\phi_{\text{CP}2} & : \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} = \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} \\
\phi_{\text{CP}3} & : \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} = \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} \\
\phi_{\text{CP}4} & : \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} = \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} \\
\phi_{\text{CP}5} & : \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix} = \begin{pmatrix} x & y & z \\ \sigma(x) & \sigma(y) & \sigma(z) \end{pmatrix}
\end{align*}
\]

The analysis is then based on the following equations:

\[
\begin{align*}
\text{CP}_{\text{entry}}(1) &= \lambda x. T \\
\text{CP}_{\text{entry}}(2) &= \text{CP}_{\text{exit}}(1) \\
\text{CP}_{\text{entry}}(3) &= \text{CP}_{\text{exit}}(2) \\
\text{CP}_{\text{entry}}(4) &= \text{CP}_{\text{exit}}(2) \\
\text{CP}_{\text{entry}}(5) &= \text{CP}_{\text{exit}}(3) \sqcup \text{CP}_{\text{exit}}(4)
\end{align*}
\]
The solutions are given by:

\[
\begin{align*}
CP_{\text{entry}}(1) &= \begin{array}{ccc} x & y & z \\
& \top & \top & \top \end{array} \\
CP_{\text{entry}}(2) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array} \\
CP_{\text{entry}}(3) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array} \\
CP_{\text{entry}}(4) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array} \\
CP_{\text{entry}}(5) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array}
\end{align*}
\]

and

\[
\begin{align*}
CP_{\text{exit}}(1) &= \begin{array}{ccc} x & y & z \\
& 1 & 2 & \top \end{array} \\
CP_{\text{exit}}(2) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array} \\
CP_{\text{exit}}(3) &= \begin{array}{ccc} x & y & z \\
& 1 & 2 & \top \end{array} \\
CP_{\text{exit}}(4) &= \begin{array}{ccc} x & y & z \\
& \top & 2 & \top \end{array} \\
CP_{\text{exit}}(5) &= \begin{array}{ccc} x & y & z \\
& \top & \top & \top \end{array}
\end{align*}
\]

Exercise 2 Consider the following simple imperative language with statements:

\[
S ::= x := a \quad \text{skip} \quad S_1 ; S_2 \quad \text{if } b \text{ then } S_1 \text{ else } S_2 \quad \text{while } b \text{ do } S
\]

Define an instance of the monotone framework (similar to the Constant Propagation Analysis CP) which performs a usage analysis Use of expressions.
For each program point, the Use Analysis will determine the minimum and maximum number of program points the value of an arithmetic expression will be used when leaving this program point and before any variables in the expressions get redefined.

Assume that labelling, flow (flow) and reverse flow (flowR), as well as initial and final labels are defined as usual. Let AExp⁎ be the set of arithmetic sub-expressions in a program S. The lattice of abstract states is given by:

\[ \hat{\text{State}} = (AExp⁎ \rightarrow N_\infty \times N_\infty) \]

with \( N_\infty = \{0, 1, 2, \ldots \} \cup \infty \). These record (at every label) the number of times an expression might be used and the number of times it is guaranteed to be used. Define a least upper bound operator \( \sqcup \) and \( \sqsubseteq \) on \( \hat{\text{State}} \) and identify the \( \perp \) and \( \top \) elements.

Use \( \hat{\text{State}} \) to define a Use Analysis like a monotone framework instance, i.e. specify the flow \( F \), the extreme labels \( E \), their initialization \( \iota \) and the transfer functions. Derive the data-flow equations for the following program (after labelling it)

\[
\begin{align*}
y &:= a \times b \\
\text{while} \ (x < a \times b) \ 	ext{do} \ \\
\ &\quad (x := a + b; \\
\ &\quad y := a + b); \\
\ &\quad x := a \times b;
\end{align*}
\]

Is this a may or a must analysis? How is it related to the VB analysis?

Discuss whether \( \hat{\text{State}} \) fulfills the Ascending/Descending Chain Conditions, and whether the Use Analysis is computable, or how it can be made computable.

Solution  Describe the property “usage of an expression” by a pair, the first component is the minimal usage number, second component is maximum usage number. Denote the projections of first and second element in a pair by \( (x, y) \mid_1 = x \) and \( (x, y) \mid_2 = y \).

Bottom:
\[
\perp = \lambda e.(\infty, 0) \text{ with } e \in AExp⁎,
\]

Top:
\[
\top = \lambda e.(0, \infty) \text{ with } e \in AExp⁎,
\]

Order:
\[
\sigma_1 \sqsubseteq \sigma_2 \text{ iff } \sigma_1(e) \mid_1 \geq \sigma_2(e) \mid_1 \text{ and } \sigma_1(e) \mid_2 \leq \sigma_2(e) \mid_2 \ \forall e \in AExp⁎,
\]

Least Upper Bound:
\[
(\sigma_1 \sqcup \sigma_2)(e) = (\min(\sigma_1(e) \mid_1, \sigma_2(e) \mid_1), \max(\sigma_1(e) \mid_2, \sigma_2(e) \mid_2)) \text{ with } e \in AExp⁎,
\]

This is a backward analysis, i.e. \( F = \text{flow}^R \).
The extremal labels $E$ are therefore $final(S)$, and the initialization is $\iota(\ell) = \lambda e.(0,0)$. Use the following notation: $(x,y) + 1 = (x+1,y+1)$ for pairs.

Transfer functions:

- $[x := a]^{\ell} : f_\ell(\sigma) = \begin{cases} f_\ell(\sigma)(e) = \sigma(e) + 1 & \text{if } e \in AExp(a) \text{ and } x \not\in FV(e) \\ f_\ell(\sigma)(e) = (0,0) & \text{if } x \in FV(e) \\ f_\ell(\sigma)(e) = \sigma(e) & \text{otherwise} \end{cases}$

- $[^{\text{skip}}]^{\ell} : f_\ell(\sigma) = \sigma$

- $[^{b}]^{\ell} : f_\ell(\sigma) = \begin{cases} f_\ell(\sigma)(e) = \sigma(e) + 1 & \text{if } e \in AExp(b) \\ f_\ell(\sigma)(e) = \sigma(e) & \text{otherwise} \end{cases}$

These are similar to the VB analysis.

Labelling:

\[
\begin{align*}
[y := a*b]^1 \\
\text{while } [(x < a * b)]^2 \text{ do } \begin{cases} \\
    [x := a+b]^3; \\
    [y := a+b]^4); \\
    [x := a*b]^5; \\
\end{cases} \\
\text{Use}_{entry}(1) = f_1(\text{Use}_{exit}(1)) \\
\text{Use}_{entry}(2) = f_2(\text{Use}_{exit}(2)) \\
\text{Use}_{entry}(3) = f_3(\text{Use}_{exit}(3)) \\
\text{Use}_{entry}(4) = f_4(\text{Use}_{exit}(4)) \\
\text{Use}_{entry}(5) = f_5(\text{Use}_{exit}(5)) \\
\end{align*}
\]

or more explicitely

\[
\begin{align*}
\text{Use}_{entry}(1) &= \text{Use}_{exit}(1)[(a * b) \mapsto (0,0)](a * b) + 1 \\
\text{Use}_{entry}(2) &= \text{Use}_{exit}(2)[(a * b) \mapsto (0,0)](a * b) + 1 \\
\text{Use}_{entry}(3) &= \text{Use}_{exit}(3)[(a + b) \mapsto (0,0)](a + b) + 1 \\
\text{Use}_{entry}(4) &= \text{Use}_{exit}(4)[(a + b) \mapsto (0,0)](a + b) + 1 \\
\text{Use}_{entry}(5) &= \text{Use}_{exit}(4)[(a * b) \mapsto (0,0)](a * b) + 1 \\
\end{align*}
\]

\[
\begin{align*}
\text{Use}_{exit}(1) &= \text{Use}_{entry}(2) \\
\text{Use}_{exit}(2) &= \text{Use}_{entry}(3) \sqcup \text{Use}_{entry}(5) \\
\text{Use}_{exit}(3) &= \text{Use}_{entry}(4) \\
\text{Use}_{exit}(4) &= \text{Use}_{entry}(2) \\
\text{Use}_{exit}(5) &= [(a * b) \mapsto (0,0), (a + b) \mapsto (0,0)] \\
\end{align*}
\]
This a monotone framework instance so may/must does not really makes sense. However, one can see it as a combination of a must and a may analysis. If the minimal number of usages is zero then we can conclude that an expression is not very busy at the exit from a program point, otherwise it is very busy.

The lattice $\widehat{\text{State}}$ does not fulfill the ACC/DCC but we only need

$$\widehat{\text{State}} = (\text{AExp}_* \rightarrow |\text{Lab}_*| \times |\text{Lab}_*|)$$

if we keep also information about which labels are already recorded and avoid counting them twice. This results in a finite lattice and thus fulfills the ACC/DCC, therefore the MFP solutions can then always be computed.

**Exercise 3** Consider the following program:

$$[x:=1]; [x:=x-1]; [x:=2]$$

Clearly $x$ is dead at the exits from 2 and 3. But $x$ is live at the exit of 1 even though its only use is to calculate a new value for a variable that turns out to be dead.

We shall say that a variable is a faint variable if it is dead or if it is only used to calculate new values for faint variables; otherwise it is strongly live.

In the example $x$ is faint at the exits from 1, 2 and 3.

Define a Data Flow Analysis that detects Strongly Live variables.

**Solution** (Sketch)

Two alternative approaches:

1. Base the analysis on the Live Variables Analysis. The function $\text{gen}_L$ must be changed to take an additional input — a set of strongly live variables:

   $$\text{gen}_L : (\text{Blocks}_* \times \mathcal{P}(\text{Var}_*)) \rightarrow \mathcal{P}(\text{Var}_*)$$

   $$\text{gen}_L([x := a], X) = \begin{cases} \text{FV}(a) & \text{if } x \in X \\ \emptyset & \text{otherwise} \end{cases}$$

   $$\text{gen}_L([\text{skip}], X) = \emptyset$$

   $$\text{gen}_L([b], X) = \text{FV}(b)$$

   and also $\text{LV}_{entry}$:

   $$\text{LV}_{entry}(\ell) = (\text{LV}_{exit}(\ell) \setminus \text{kill}_{\text{LV}}(B^\ell)) \cup \text{gen}_L(B^\ell, \text{LV}_{exit}(\ell))$$

2. As a monotone framework with transfer functions:

   $$f_{[x := a]}(X) = \begin{cases} (X \setminus \{x\}) \cup \text{FV}(a) & \text{if } x \in X \\ X & \text{otherwise} \end{cases}$$

   $$f_{[\text{skip}]}(X) = X$$

   $$f_{[b]}(X) = X \cup \text{FV}(b)$$