Exercises

Program Analysis (CO70020)

Sheet 1

Exercise 1 *Give a labelling of the following program and the (intuitive) flow 'flow' and the reverse flow 'flow^R'.*

```
x := 1;
while y>0  do (
           if y<=0
           then       x := x+3
           else       skip
           x := x-1;
           z := z+x;
           )
x := 2;
```

What "simplifications" could you think of with regard to the guard $y > 0$? What happens if you generalise your approach to any guard/test predicate $p(y)$ (and not $p(y)$, respectively)?

Solution Obviously the **if** is redundant. Nothing is executed in the **if** statement as long as the **while** -loop is executed (only **skip**).

But if $p(y)$ is undecided then the program may not terminate (because the evaluation of $p(y)$ may not terminate). In this case all judgments about the following statements are vacuous. This is especially relevant in a concurrent context.

Exercise 2 *Consider the following* **While** *program:*

```
x:=1;
if (x>0)
then x:=x-1
else y:=y-1
```

Construct the flow formally.

Solution

1. Label program

$$[\texttt{x:=1}]^1;\ \textbf{if}\ ([\texttt{x>0}]^2)\ \textbf{then}\ [\texttt{x:=x-1}]^3\ \textbf{else}\ [\texttt{y:=y-1}]^4$$

2. Construct flow

$$
\begin{aligned}
&\mathit{flow}([\texttt{x:=1}]^1;\ \textbf{if}\ \ ([\texttt{x>0}]^2)\ \textbf{then}\ \ [\texttt{x:=x-1}]^3\ \textbf{else}\ \ [\texttt{y:=y-1}]^4) = \\
&= \mathit{flow}([\texttt{x:=1}]^1) \cup \mathit{flow}(\textbf{if}\ \ ([\texttt{x>0}]^2)\ \textbf{then}\ \ [\texttt{x:=x-1}]^3\ \textbf{else}\ \ [\texttt{y:=y-1}]^4) \\
&\quad \cup\{(\ell, \mathit{init}(\textbf{if}\ \ ([\texttt{x>0}]^2)\ \textbf{then}\ \ [\texttt{x:=x-1}]^3\ \textbf{else}\ \ [\texttt{y:=y-1}]^4)) \mid \ell \in \mathit{final}([\texttt{x:=1}]^1\} = \\
&= \emptyset \cup \mathit{flow}(\textbf{if}\ \ ([\texttt{x>0}]^2)\ \textbf{then}\ \ [\texttt{x:=x-1}]^3\ \textbf{else}\ \ [\texttt{y:=y-1}]^4) \cup \{(\ell, 2) \mid \ell \in \{1\}\} = \\
&= \mathit{flow}(\textbf{if}\ \ ([\texttt{x>0}]^2\ \textbf{then}\ \ [\texttt{x:=x-1}]^3\ \textbf{else}\ \ [\texttt{y:=y-1}]^4) \cup \{(1,2)\} = \\
&= \mathit{flow}([\texttt{x:=x-1}]^3) \cup \mathit{flow}([\texttt{y:=y-1}]^4) \cup \{(2, \mathit{init}([\texttt{x:=x-1}]^3)), (2, \mathit{init}([\texttt{y:=y-1}]^4))\} \cup \{(1,2)\} = \\
&= \emptyset \cup \emptyset \cup \{(2,3),(2,4)\} \cup \{(1,2)\} = \\
&= \{(1,2),(2,3),(2,4)\}
\end{aligned}
$$

Exercise 3 *Guess the* RD *solutions for the following three* **While** *programs:*

```
x := 4;
z := 2;
if y > x then
   x := 3;
else
   x := 4;
z := x;
```

```
x := 4;
z := 2;
if y > x then
   x := 3;
else
   x := 3;
z := x;
```

```
x := 4;
y := 2;
if y > x then
   x := 3;
else
   x := 5;
z := x;
```

What kind of optimisation could you suggest.

Solution

First program: The program points could be labelled as follows (similar for the other two):

```
[x := 4;]^1
[z := 2;]^2
if [y > x]^3 then
   [x := 3;]^4
else
   [x := 4;]^5
[z := x;]^6
```

then one could expect perhaps the following RD solutions:

$$
\begin{aligned}
\mathsf{RD}_{entry}(1) &= \{(\mathrm{x},?),(\mathrm{y},?),(\mathrm{z},?)\} \\
\mathsf{RD}_{exit}(1) &= \{(\mathrm{x},1),(\mathrm{y},?),(\mathrm{z},?)\} \\
\mathsf{RD}_{entry}(2) &= \{(\mathrm{x},1),(\mathrm{y},?),(\mathrm{z},?)\} \\
\mathsf{RD}_{exit}(2) &= \{(\mathrm{x},1),(\mathrm{y},?),(\mathrm{z},2)\} \\
\mathsf{RD}_{entry}(3) &= \{(\mathrm{x},1),(\mathrm{y},?),(\mathrm{z},2)\}
\end{aligned}
$$

$$\begin{array}{rcl}
\mathsf{RD}_{exit}(3) & = & \{(\mathtt{x},1),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{entry}(4) & = & \{(\mathtt{x},1),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{exit}(4) & =, & \{(\mathtt{x},4),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{entry}(5) & = & \{(\mathtt{x},1),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{exit}(5) & = & \{(\mathtt{x},5),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{entry}(6) & = & \{(\mathtt{x},4),(\mathtt{x},5),(\mathtt{y},?),(\mathtt{z},2)\} \\
\mathsf{RD}_{exit}(6) & = & \{(\mathtt{x},4),(\mathtt{x},5),(\mathtt{y},?),(\mathtt{z},6)\}
\end{array}$$

The RD analysis suggest that it could be possible (using e.g. the Constant Folding rewrite/program transformation from the introduction) to simplify the test at label 3:

```
[x := 4;]^1
[z := 2;]^2
if [y > 4]^3 then
   [x := 3;]^4
else
   [x := 4;]^5
[z := x;]^6
```

as only $(\mathtt{x},1)$ reaches label 3 (see $\mathsf{RD}_{entry}(3)$).

In itself the program transformation from the introduction would not allow however to eliminate statement 1 in order to get:

```
[z := 2;]^2
if [y > 4]^3 then
   [x := 3;]^4
else
   [x := 4;]^5
[z := x;]^6
```

as this might lead to a non-initialised variable `x`. Nevertheless, one could argue that this non-initialisation issue is not relevant as any continuation of this program fragment would be provided with an initialised variable `x` (as $\mathsf{RD}_{exit}(6)$ does not contain $(\mathtt{x},?)$).

Another simplification – for which however the information the RD analysis provides is not sufficent (an additional analysis would be needed) – could produce even:

```
if [y > 4]^3 then
   [z := 3;]^4
else
   [z := 4;]^5
```

However, this eliminates `x` completely, which is only justified if this variable is not needed at any later stage.

Note: Except for the first kind of optimisation we could not simplify the program any furher in a concurrent context, i.e. if this fragment specifies a 'thread' which is executed in parallel with some others (using shared variables).

Second program: The solutions for RD should be the same as in the previous example (as both programs have the same control flow structure).

The constant folding transformations presented in the lecture allow for a replacement of the last statement by $[\texttt{z := 3;}]^6$.

However, one might also argue that the whole `if`-statement could be replaced by $[\texttt{x := 3;}]^3$ or perhaps even eliminated completely (if `x` is not "used later"). But these optimisations are not supported formally.

Third program: Again, the solutions for RD should be essentially the same as in the previous examples (as all programs have the same control flow structure, but slightly different local transfer functions).

No optimisation using constant folding is justified as in the first example. However, we know that the test in the `if`-statement will always fail (as $2 > 4$ is false) thus `x` will always be 5 when we reach the last statement. Thus the program could be replaced perhaps by `y:=2; x:=5; z:=5;`.

However, this kind of optimisation is not what we aim for in static program analysis as it requires a general way to decide when tests (in `if`-statements) succeed or fail, but this is in general undecidable and would lead to eventually non-terminating 'optimisation' strategies.

Exercise 4 *Construct the* RD *equations for the following program:*

```
x := 4;
z := 2;
if y > x then
   x := 3;
else
   x := 4;
z := x;
```

Solution

1. The program points c(s)ould be labelled as follows:

$[\texttt{x := 4;}]^1$
$[\texttt{z := 2;}]^2$
$\texttt{if } [\texttt{y > x}]^3 \texttt{ then}$
$\quad [\texttt{x := 3;}]^4$

```
else
   [x := 4;]^5
[z := x;]^6
```

2. Construct the flow

$$\mathit{flow}(\ldots) = \{(1,2),(2,3),(3,4),(3,5),(4,6),(5,6)\}$$

3. Construct local $\mathit{kill}_{\mathsf{RD}}$ and $\mathit{gen}_{\mathsf{RD}}$

$$\begin{array}{lcl}
\mathit{gen}_{\mathsf{RD}}([\ldots]^1) & = & \{(\mathtt{x},1)\} \\
\mathit{gen}_{\mathsf{RD}}([\ldots]^2) & = & \{(\mathtt{z},2)\} \\
\mathit{gen}_{\mathsf{RD}}([\ldots]^3) & = & \emptyset \\
\mathit{gen}_{\mathsf{RD}}([\ldots]^4) & = & \{(\mathtt{x},4)\} \\
\mathit{gen}_{\mathsf{RD}}([\ldots]^5) & = & \{(\mathtt{x},5)\} \\
\mathit{gen}_{\mathsf{RD}}([\ldots]^6) & = & \{(\mathtt{z},6)\}
\end{array}$$

$$\begin{array}{lcl}
\mathit{kill}_{\mathsf{RD}}([\ldots]^1) & = & \{(\mathtt{x},?),(\mathtt{x},4),(\mathtt{x},5),(\mathtt{x},1)\} \\
\mathit{kill}_{\mathsf{RD}}([\ldots]^2) & = & \{(\mathtt{z},?),(\mathtt{z},2),(\mathtt{z},6)\} \\
\mathit{kill}_{\mathsf{RD}}([\ldots]^3) & = & \emptyset \\
\mathit{kill}_{\mathsf{RD}}([\ldots]^4) & = & \{(\mathtt{x},?),(\mathtt{x},4),(\mathtt{x},5),(\mathtt{x},1)\} \\
\mathit{kill}_{\mathsf{RD}}([\ldots]^5) & = & \{(\mathtt{x},?),(\mathtt{x},4),(\mathtt{x},5),(\mathtt{x},1)\} \\
\mathit{kill}_{\mathsf{RD}}([\ldots]^6) & = & \{(\mathtt{z},?),(\mathtt{z},2),(\mathtt{z},6)\}
\end{array}$$

4. State RD equations:

$$\begin{array}{lcl}
\mathsf{RD}_{entry}(1) & = & \{(\mathtt{x},?),(\mathtt{y},?),(\mathtt{z},?)\} \\
\mathsf{RD}_{entry}(2) & = & \mathsf{RD}_{exit}(1) \\
\mathsf{RD}_{entry}(3) & = & \mathsf{RD}_{exit}(2) \\
\mathsf{RD}_{entry}(4) & = & \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{entry}(5) & = & \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{entry}(6) & = & \mathsf{RD}_{exit}(4) \cup \mathsf{RD}_{exit}(5)
\end{array}$$

$$\begin{array}{lcl}
\mathsf{RD}_{exit}(1) & = & \mathsf{RD}_{entry}(1) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^1) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^1) \\
\mathsf{RD}_{exit}(2) & = & \mathsf{RD}_{entry}(2) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^2) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^2) \\
\mathsf{RD}_{exit}(3) & = & \mathsf{RD}_{entry}(3) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^3) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^3) \\
\mathsf{RD}_{exit}(4) & = & \mathsf{RD}_{entry}(4) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^4) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^4) \\
\mathsf{RD}_{exit}(5) & = & \mathsf{RD}_{entry}(5) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^5) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^5) \\
\mathsf{RD}_{exit}(6) & = & \mathsf{RD}_{entry}(6) \setminus \mathit{kill}_{\mathsf{RD}}([\ldots]^6) \cup \mathit{gen}_{\mathsf{RD}}([\ldots]^6)
\end{array}$$

Fill in values for $\mathit{kill}_{\mathsf{RD}}$ and genRD from above.

Exercise 5 *Is there a program such that:*

1. $\{(\mathtt{x},1),(\mathtt{x},4),(\mathtt{x},8)\} \subseteq \mathsf{RD}_{entry}(9)$, *or a program such that:*
2. $\{(\mathtt{x},1),(\mathtt{x},4),(\mathtt{y},4)\} \subseteq \mathsf{RD}_{entry}(9)$

Give example(s) or argument(s).

Solution For example of a program with $\{(\mathtt{x},1),(\mathtt{x},4),(\mathtt{x},8)\} \subseteq \mathsf{RD}_{entry}(9)$:

```
[x := ...;]^1
[y := ...;]^2
if [y > 2]^3 then
   [x := ...;]^4
else
   [y := ...;]^5
if [z <> 2]^6 then
   [y := ...;]^7
else
   [x := ...;]^8
[z := ...;]^9
```

Arguably, there is no (**while** !) program of the second kind as at each location or program point only one variable can be defined (assigned to).