Data Flow Assignment I
Automatic Program Analysis

Chris Eidhof, Rui S. Barbosa

February 21, 2009

1

Our analysis for Strongly Live Variables is almost the same as the Live Variables analysis. We now only present the changes made.

Below is the definition of the gen_{SLV} and $kill_{SLV}$ functions. Note that we changed the gen function to take an extra argument l of type $L = \mathcal{P}(\mathbf{Var}_\star)$ which corresponds to the set of strongly live variables at the exit of the corresponding block. This is needed because the strong liveliness of a variable before being used in an assignment block depends on whether the assigned variable is strongly live at the exit of that block.

$$
\begin{aligned}
kill_{SLV} &: \mathbf{Blocks}_\star \to \mathcal{P}(\mathbf{Var}_\star) \\
kill_{SLV}([x := a]^\ell) &= \{x\} \\
kill_{SLV}([\texttt{skip}\,]^\ell) &= \emptyset \\
kill_{SLV}([b]^\ell) &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
gen_{SLV} &: \mathbf{Blocks}_\star \times \mathcal{P}(\mathbf{Var}_\star) \to \mathcal{P}(\mathbf{Var}_\star) \\
gen_{SLV}([x := a]^\ell, l) &= \begin{cases} FV(a) & \text{if } x \in l \\ \emptyset & \text{otherwise} \end{cases} \\
gen_{SLV}([\texttt{skip}\,]^\ell, l) &= \emptyset \\
gen_{SLV}([b]^\ell, l) &= FV(b)
\end{aligned}
$$

This means our f_ℓ also has to change too:

$$f_\ell(l) = (l \setminus kill([B]^\ell)) \cup gen([B]^\ell, l) \text{ where } [B]^\ell \in blocks(S_\star)$$

Moreover, in our analysis, ι will not be necessarily empty. Instead, it will be a chosen set of variables of interest. By now, there are no means for a program to communicate

its results besides the inspection of some variables when execution halts: those variables are what we call the variables of interest at the end of the program. Clearly, those variables need to be considered live at the exit of all final blocks, so that we can always inspect their values. When ι is empty, the only way to have intermediate non-empty sets of strongly live variables will be by using them in conditional expressions (which control the flow of the program). In that case, we are not interested in any output from the program. As we shall see, the print statement to be introduced later will change this situation, as it will allow for a program to communicate to the outside at any point of execution.

2

We developed some modules in HASKELL which allow us to define and perform data flow analysis on WHILE programs. To define an analysis, one needs to specify how to create a monotone framework from a program. For the flow graph generation, one can use the functions `forward` and `backward`. As for the transfer functions, also a few useful combinators were introduced, mainly for the cases when the underlying lattice is a set: `may` and `must` can be used to specify which operation to use and there is also special support for the common $gen/kill$ transfer functions (`genkill`) and for $gen/kill$ functions that can receive an additional parameter as in the case of SLV (`depgenkill`). Having defined the conversion from programs to monotone frameworks, our modules automaticaly generate the equations, solve them through chaotic iteration and group the results in a table written to TEX format.

As an example, the definition of SLV will read as:

```
stronglivevariables i  =
       createDataFlowAnalyser
         backward
         (may (const i,const( depgenkill(genSLV,const.killSLV) ) ))
```

where the `genSLV` and `killSLV` are just as defined in the previous part. Then, we just define

```
doSLV iota = (resultToTable ("SLV",True,"exit","entry") .
              scan_analyze (stronglivevariables S.empty) . labelProgram)
```

which is a function that given a set of variables of interest and an unlabeled program, performs the strong live variable analysis and displays the intermediate results in a TEX table.

To demonstrate the Strongly Live Variable Analysis, we will use this HASKELLprogram to perform chaotic iteration on the following simple example program

Program 1.

$[r := 1]^1$;
$[a := r * r]^2$;
`while` $[y > 0]^3$;
  $[r := r * x]^4$;
  $[t := y]^5$;
  $[y := t - 1]^6$;
$[\texttt{skip}]^7$

Tables 1, 2, 3 and 4 display a trace of the chaotic iteration algorithm for several different values for ι. The last two columns are always repeated, indicating that the process has stabilized, reaching a fix point. We have the guarantee that this situation always happens eventually. For comparison purposes, we have also used our module to perform (regular) Live Variable Analysis on the same program, obtaining the results in Table 5.

We can easily observe that the results of the SLV analysis significantly vary depending on ι. If we are interested in knowing the value of r in the end (by the way, that would be the power function, computing x^y), then the result of the analysis is just that of simple life variable analysis. One can observe that both analysis detect the indirection through t when decrementing the value of y: *between* labels 5 and 6, both analysis detect that y is dead and t is alive instead. On the opposite side, let us look more closely to the results for $\iota = \emptyset$. As we have already said, this can be thought of as if we do not require any response from the program. Being so, one could safely ignore all the assignments which are only used to produce results and do not interfere (not even inderectly) with the execution flow control. The only variables that need to be considered live at a point are those which carry values that will eventually be used to determine the trace of execution (that will be used in a while or if condition). As we can observe, the SLV analysis is able to detect that only the variable y (and, between statements 5 and 6, the variable t) will be ever used to determine the flow of the program. Actually, the number of iterations of the while loop that will be done depends only on the value of y at the start of the program (thus $SLV_{entry}(1) = \{y\}$). The regular live variable analysis is not able to detect this. For example, the variable x will be live at the entry of the while loop, because its value will be used to change the value of r in the execution of the body. However, it is a faint variables in this case, because the value of r will never be used: when the end of the program is reached, its value is just forgotten without having ever been used except in assignments to other variables (a and r itself) which are just faint or dead themselves. The other cases for ι are similar and we will not discuss them.

Table 1: Strongly Live Variable Analysis on Program 1 with $\iota = \{r\}$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{x,y\}$	$\{x,y\}$	$\{x,y\}$
$SLV_{exit}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{exit}(2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{exit}(3)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{exit}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{exit}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$
$SLV_{entry}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$
$SLV_{exit}(6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(7)$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$
$SLV_{exit}(7)$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$

Table 2: Strongly Live Variable Analysis on Program 1 with $\iota = \{y\}$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (3)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$
$SLV_{entry}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$
$SLV_{exit}\ (6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(7)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (7)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$

Table 3: Strongly Live Variable Analysis on Program 1 with $\iota = \{a\}$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$
$SLV_{exit}\ (2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{exit}\ (3)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{exit}\ (4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{exit}\ (5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$
$SLV_{entry}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$	$\{a,t\}$
$SLV_{exit}\ (6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$	$\{a,y\}$
$SLV_{entry}(7)$	$\emptyset$	$\emptyset$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$
$SLV_{exit}\ (7)$	$\emptyset$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$	$\{a\}$

Table 4: Strongly Live Variable Analysis on Program 1 with $\iota = \emptyset$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (3)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{exit}\ (5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$
$SLV_{entry}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$
$SLV_{exit}\ (6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$
$SLV_{entry}(7)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$SLV_{exit}\ (7)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 5: Live Variable Analysis on Program 1

$LV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{x,y\}$	$\{x,y\}$	$\{x,y\}$
$LV_{exit}\ (1)$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{entry}(2)$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{exit}\ (2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{exit}\ (3)$	$\emptyset$	$\emptyset$	$\{r,x\}$	$\{r,x\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{entry}(4)$	$\emptyset$	$\{r,x\}$	$\{r,x\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{exit}\ (4)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{entry}(5)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{exit}\ (5)$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$
$LV_{entry}(6)$	$\emptyset$	$\{t\}$	$\{t\}$	$\{t\}$	$\{t\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$	$\{r,t,x\}$
$LV_{exit}\ (6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$LV_{entry}(7)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$LV_{exit}\ (7)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

3

We start by adding the new labelled constructs to the abstract syntax of the language:

$$\begin{aligned} Stmt ::= \ & \ldots \\ & \mid [\texttt{print}\ a]^\ell \\ & \mid [x_1, \ldots, x_n := a_1, \ldots, a_n]^\ell, n \in \mathbb{N} \\ & \mid [\texttt{continue}\ ]^\ell \\ & \mid [\texttt{break}\ ]^\ell \end{aligned}$$

We also extend the $blocks$ function accordingly in a pretty straightforward manner. The definition of the $labels$ function given in the book stays valid.

$$\begin{aligned} blocks([\texttt{print}\ a]^\ell) &= \{[\texttt{print}\ a]^\ell\} \\ blocks([x_1, \ldots, x_n := a_1, \ldots, a_n]^\ell) &= \{[x_1, \ldots, x_n := a_1, \ldots, a_n]^\ell\} \\ blocks([\texttt{continue}\ ]^\ell) &= \{[\texttt{continue}\ ]^\ell\} \\ blocks([\texttt{break}\ ]^\ell) &= \{[\texttt{break}\ ]^\ell\} \end{aligned}$$

Then, for each of those constructs, we shall give a description of its semantics (the actual rules may be found in table 3.3) and explain the modification that need to be done to the monotone framework, particularly to the functions defining the flow control ($init$, $final$ and $flow$) as well as to the gen_{SLV} and $kill_{SLV}$ functions which define our analysis. We also give some example programs in order to demonstrate the resulting analysis.

3.1 Print

Informally, the semantics of this construct will be to write the value of arithmetic expression a to an output stream. When defining the formal semantics, we will add an extra component to the state corresponding to the list of values printed so far ($out \in Z^\star$). The new state will then be $\sigma = (\eta, out) \in (Var \to \mathbb{Z}) \times \mathbb{Z}^\star$ and the old rules will stay valid if we interpret $\sigma[x \mapsto y] = (\eta, out)[x \mapsto y]$ as $(\eta[x \mapsto y], out)$.

This new construct has no effect on the flow control of the program. Hence, the related functions are defined just as for the other simple statements (assignments and `skip`).

$$\begin{aligned} init([\texttt{print}\ a]^\ell) &= \ell \\ final([\texttt{print}\ a]^\ell) &= \{\ell\} \\ flow([\texttt{print}\ a]^\ell) &= \emptyset \end{aligned}$$

From the Strongly Live Variables Analysis point of view, what is relevant is the added constraint that all the variables used in expression a need to be considered (strongly)

live when entering the $[\texttt{print } a]^\ell$ block. Hence, the gen and $kill$ are extended in this way:

$$kill_{SLV}([\texttt{print } a]^\ell) = \emptyset$$

$$gen_{SLV}([\texttt{print } a]^\ell, l) = FV(a)$$

Note that the `print` construct provides the program with a new capability of conveying results to the outside. Thus, we are not required to consider $\iota \neq \emptyset$ from now on. We simply need to print the so called variables of interest at the end of our program. Of course, we can do that anywhere else, thus giving the possibility of inserting variables of interest at any point. For example, replacing the `skip` by $[\texttt{print } r]$ at the end of program 1 (obtaining Program 2), we get the results in Table 6. Those are (almost) the same as in Table 1, the only difference being that, after the `print` statement, r is not alive in this case (which is clearly expectable).

Program 2.

$[r := 1]^1$;
$[a := r * r]^2$;
`while` $[y > 0]^3$;
  $[r := r * x]^4$;
  $[t := y]^5$;
  $[y := t - 1]^6$;
$[\texttt{print}r]^7$

3.2 Simultaneous Assignements

The meaning of multiple assignments ($[v_1, \ldots, v_n := a_1, \ldots, a_n]^\ell$) is pretty straightforward: all expressions in the right hand side are evaluated and then each of them is assigned to the corresponding variable on the left hand side, in left to right order.

As in the previous case, the flow related functions are easily extended.

$$\begin{aligned} init([v_1, \ldots, v_n = a_1, \ldots, a_n]^\ell) &= \ell \\ final([v_1, \ldots, v_n = a_1, \ldots, a_n]^\ell) &= \{\ell\} \\ flow([v_1, \ldots, v_n = a_1, \ldots, a_n]^\ell) &= \emptyset \end{aligned}$$

Clearly, as with simple assignements, the $kill$ function just enumerates the variables which are assigned to in ℓ.

On the other hand, the gen function needs to give all the variables required to be strongly live before the block. As a first guess, we could include all the free variables

Table 6: Strongly Live Variable Analysis on Program 2 (print statement) with $\iota = \emptyset$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{y\}$	$\{x, y\}$	$\{x, y\}$	$\{x, y\}$
$SLV_{exit}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{exit}(2)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{entry}(3)$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{exit}(3)$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{exit}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{exit}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t, x\}$	$\{r, t, x\}$	$\{r, t, x\}$	$\{r, t, x\}$
$SLV_{entry}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{t\}$	$\{t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t\}$	$\{r, t, x\}$	$\{r, t, x\}$	$\{r, t, x\}$	$\{r, t, x\}$	$\{r, t, x\}$
$SLV_{exit}(6)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$	$\{r, x, y\}$
$SLV_{entry}(7)$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$
$SLV_{exit}(7)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

used in expressions assigned to variables which are strongly live after the block, closely following the single assignement case. However, this solution is not optimal in the case where the v_i are not pairwise distinct. For example, let us consider the block $[x, y, x := a, b, c]^\ell$) and say x and y are strongly live after it. Then, the variables in expression a need not be considered strongly live before the block, as the attribution $x := a$ will be immediately overwritten by $x := c$ (recall that attributions are done in left to right order).

Therefore, we extend the $kill$ and gen functions in the following manner:

$$kill_{SLV}([v_1, \ldots, v_n := a_1, \ldots, a_n]^\ell) = \{v_i \mid 1 \leq i \leq n\}$$

$$gen_{SLV}([v_1, \ldots, v_n := a_1, \ldots, a_n]^\ell, l) = \bigcup\{FV(a_i) \mid v_i \in l \wedge \neg\exists j : j > i \,.\, v_j = v_i\}$$

To demonstrate the use of this construct, let us again consider a slightly modified version of Program 2, where the assignments inside the loop have been grouped togheter. Also, a new (dummy) assignment was added to illustrate the situation when multiple assignments to the same variable occur.

Program 3.

$[r := 1]^1$;
$[a := r * r]^2$;
`while` $[y > 0]^3$;
  $[y, r, y := a + 1, r * x, y - 1]^4$;
$[\texttt{print}r]^5$

Performing Strongly Live Variable Analysis on this program yields the results on Table 7. We can easily notice that, although a is used in an expression which is attributed to variable y (which is strongly live after the assignment), it is not considered live before that multiple assignment because y appears twice in the assigned variables and the second (rightmost) assignment overrides the one that depends on the value of a.

3.3 Break and Continue

The meaning of these constructs inside while loops is just what we are used to: a `break` statement cause the program to jump immediately out of the loop whilst a `continue` statement *restarts* the while loop execution (including testing the condition). If the constructs are found outside of a while loop, the compiler/interpreter should give an error before doing the analysis (it is a simple syntactic check).

The formal semantics could be given in a way resembling exception handling in Java (where while loops take the rôle of catch), as we briefly explain. These constructs would

Table 7: Strongly Live Variable Analysis on Program 3(multiple assignments) with $\iota = \emptyset$

$SLV_{entry}(1)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{x,y\}$
SLV_{exit} (1)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
SLV_{exit} (2)	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(3)$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
SLV_{exit} (3)	$\emptyset$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
$SLV_{entry}(4)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$	$\{r,x,y\}$
SLV_{exit} (4)	$\emptyset$	$\emptyset$	$\emptyset$	$\{y\}$	$\{y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r,y\}$	$\{r\}$
$SLV_{entry}(5)$	$\emptyset$	$\emptyset$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\{r\}$	$\emptyset$
SLV_{exit} (5)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

cause execution to halt in a new kind of state enclosing the information of the *then current* state along with an annotation of the reason for halting (`break` or `continue`). We would then need several rules to deal with those halting states. Firstly, we would need to propagate the halting states unchanged through consecutive statements: sequencing (;) with another statement would ignore that second statement and enclosing if-then-else constructs would also propagate the halting state (for the latter situation, the current rule suffices). Finally, the way to deal with `while` should also change. A possibility is to change the $[wh_1]$ rule so that it takes a look similar to that of $[seq_1]$. Then, if the body statement results in a `continue`-annotated state, the annotation is dropped and the while loop restarted. If halting state of the body is `break`-annotated, the annotation is also dropped but the execution of the while loop would be considered finished.

It is worth noticing that if a computation ends in an annotated state and there is no enclosing while loop, the program itself as a whole would end in that kind of state. We will consider that to be a error terminating program. Our framework will not consider the top-level `break` and `continue` statements to be part of the final statements of a program. That means that we do not care about the values of the variables of interest if the program terminates in that way. This should cause no harm because, as noticed above, a program with top-level `break` or `continue` statements could be easily ruled out as invalid. On the other hand, this assumption will facilitate the definition of the flow related functions: for example, as we consider that a `continue` statement is not a final statement of S_1, then the definition of $flow(S_1; S_2)$ does not consider the flow from the `continue` statement to $init(S_2)$, just like one would expect. We then need to account for the existence of these constructs in the definition of the flow information functions for `while` statements (as well as the `break` and `continue`, of course). Those definitions now read:

$$\begin{array}{rcl} init([\texttt{continue}]^\ell) & = & \ell \\ final([\texttt{continue}]^\ell) & = & \emptyset \\ flow([\texttt{continue}]^\ell) & = & \emptyset \end{array} \qquad \begin{array}{rcl} init([\texttt{break}]^\ell) & = & \ell \\ final([\texttt{break}]^\ell) & = & \emptyset \\ flow([\texttt{break}]^\ell) & = & \emptyset \end{array}$$

$$\begin{array}{rcl} init(\texttt{while}\ [b]^\ell S) & = & \ell \\ final(\texttt{while}\ [b]^\ell S) & = & \ell \cup breaksOf(S) \\ flow(\texttt{while}\ [b]^\ell S) & = & \{(l, init(S))\} \cup flow(S) \\ & & \cup \{(l', l) | l' \in final(S)\} \\ & & \cup \{(l', l) | l' \in continuesOf(S)\} \end{array}$$

where $continuesOf$ and $breaksOf$ functions are auxiliary functions which map a given program statement to the labels of its top level (not nested in a `while`) `continue` and `break` statements, respectively.

$$
\begin{array}{rcl}
\mathit{continuesOf} & : & \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab}) \\
\mathit{continuesOf}([x := a]^\ell) & = & \emptyset \\
\mathit{continuesOf}([\texttt{skip}\,]^\ell) & = & \emptyset \\
\mathit{continuesOf}([x_1, \ldots, x_\vdots = a_1, \ldots, a_n]^\ell) & = & \emptyset \\
\mathit{continuesOf}([\texttt{print}\ a]^\ell) & = & \emptyset \\
\mathit{continuesOf}([\texttt{continue}\,]^\ell) & = & \{\ell\} \\
\mathit{continuesOf}([\texttt{break}\,]^\ell) & = & \emptyset \\
\mathit{continuesOf}(S_1; S_2) & = & \mathit{continuesOf}(S_1) \cup \mathit{continuesOf}(S_2) \\
\mathit{continuesOf}(\texttt{if}\ [b]^\ell \texttt{then}\ S_1 \texttt{else}\ S_2) & = & \mathit{continuesOf}(S_1) \cup \mathit{continuesOf}(S_2) \\
\mathit{continuesOf}(\texttt{while}\ [b]^\ell S) & = & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{breaksOf} & : & \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab}) \\
\mathit{breaksOf}([x := a]^\ell) & = & \emptyset \\
\mathit{breaksOf}([\texttt{skip}\,]^\ell) & = & \emptyset \\
\mathit{breaksOf}([x_1, \ldots, x_\vdots = a_1, \ldots, a_n]^\ell) & = & \emptyset \\
\mathit{breaksOf}([\texttt{print}\ a]^\ell) & = & \emptyset \\
\mathit{breaksOf}([\texttt{continue}\,]^\ell) & = & \emptyset \\
\mathit{breaksOf}([\texttt{break}\,]^\ell) & = & \{\ell\} \\
\mathit{breaksOf}(S_1; S_2) & = & \mathit{breaksOf}(S_1) \cup \mathit{breaksOf}(S_2) \\
\mathit{breaksOf}(\texttt{if}\ [b]^\ell \texttt{then}\ S_1 \texttt{else}\ S_2) & = & \mathit{breaksOf}(S_1) \cup \mathit{breaksOf}(S_2) \\
\mathit{breaksOf}(\texttt{while}\ [b]^\ell S) & = & \emptyset
\end{array}
$$

We presented above the changes made on the definition of the flow of the program. These definitions can be used in all monotone frameworks. Regarding the Strongly Live Variable analysis, these constructs are easy to handle. They are just like `skip` as they do not change the state of the data whatsoever: according to the semantics informally presented above (cf. the formal rules in Table 3.3), the state was frozen with an annotation and then de-annotated (thus recovered) when execution restarts at the right program point. Therefore, we have:

$$
\mathit{kill}_{SLV}([\texttt{continue}\,]^\ell) = \mathit{kill}_{SLV}([\texttt{break}\,]^\ell) = \emptyset
$$

$$
\mathit{gen}_{SLV}([\texttt{continue}\,]^\ell, l) = \mathit{gen}_{SLV}([\texttt{break}\,]^\ell, l) = \emptyset
$$

We can now analyze two example programs. First, we will do Strongly Live Variable

Analysis on the following program:

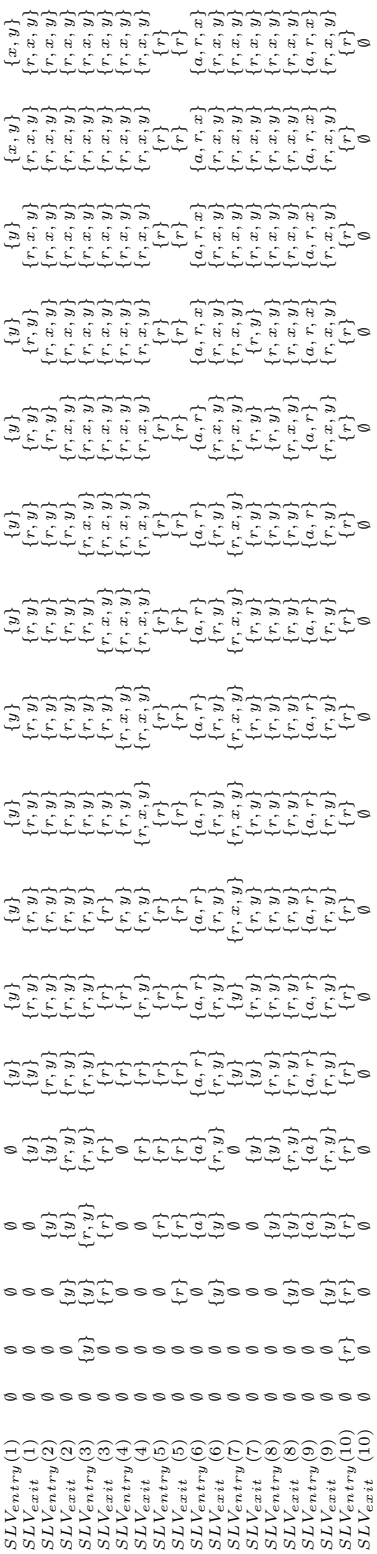
Program 4.

$[r := 1]^1$;
$[a := r * r]^2$;
`while` $[y > 1]^3$;
`if`$[y > 10]^4$
`then`
$[$`break`$]^5$;
$[y := a * a]^6$
`else`
$[r, y := r * x, y - 1]^7$;
$[$`continue`$]^8$;
$[y := a * a]^9$;
$[$`print`$r]^{10}$

The results are in Table 8. First of all, we can notice that there are certain unacessible program points (namely those immediately after the `break` and the `continue` statements, 6 and 9). Actually, there are values computed for those points as well (which are not empty, meaning they get modified during chaotic iteration). This happens because there is a path from those statements to the while condition. So, as we are performing a backward analysis, information goes from the while condition to those points. However, as these are inacessible points, there is no way that information can go from there to any acessible part of the program and contaminate the information there (being joined). So, the results we compute for that points are not actually relevant to the whole program. Those values are actually correct in the sense that if we could point to an arbitrary label and start executing there, then that those values would be correct for those points. A last remark about this: one could think a forward analysis would cause problems as information would flow from those points and be joined with the information at accessible points of the program. However, recall that the initial values at these inaccessible points would be bottom (or top) and would remain so because information would never flow to these points. Then, we would be joining with bottom (meeting with top) at the accessible points, which amounts to nothing. In the example program, one can observe that information from that points did not contaminate the rest by realizing that a is never live in accessible points although it is used in an assignment to y in those inaccessible points.

We will also consider a second program, on which we will perform Available Expression Analyses (again using our HASKELL modules).

Table 8: Strongly Live Variable Analysis on Program 4 (with `break` and `continue`)



Program 5.

```
[y := x * x]^1;
while [x > 0]^2;
  [x := x - 1]^3;
  if[y > 1024]^4
  then
    [break]^5
  else
    [skip]^6 ;
  [y := x * x]^7;
  [continue]^8;
  [x := 1]^9;
[skip]^10
```

The results of this analysis can be found in Table 9. We can easly check that the values were correctly computed. The only non-trivial expression in program 5 is $x * x$. So that is the only one that can ever appear. Now, in the inacessible program point (9), we always have $\top$ (this is a must analysis, so join is intersection). We can also see that the `continue` statement was well captured by the analysis because expression $x * x$ is always available at point 2 (either it comes from before the while, or before the continue). If the continue was ignored, that expression would have been destroyed at point 9. As for the `break` statement, we can see that altough expression $x * x$ would be passed to point 10 from point 2, as it is not available before the break, it is not necessarily available at point 10 (the break goes directly there).

Table 9: Available Expression Analysis on Program 5 (with `break` and `continue`)

$AE_{entry}(1)$	$\top$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
AE_{exit} (1)	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(2)$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (2)	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(3)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (3)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(4)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (4)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(5)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (5)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(6)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (6)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(7)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (7)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(8)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (8)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(9)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
AE_{exit} (9)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\{x*x\}$	$\{x*x\}$	$\{x*x\}$
$AE_{entry}(10)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\emptyset$	$\emptyset$
AE_{exit} (10)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\emptyset$	$\emptyset$

Table 10: Modifications to the Operational Semantics

state	$\sigma = (\eta, out) \in (Var \to \mathbb{Z}) \times \mathbb{Z}^\star$
$[print]$	$\langle \texttt{print}\ a, \sigma = (\eta, out)\rangle \longrightarrow (\eta, snoc(out, \mathcal{A}[\![a]\!])\eta))$
$[multass]$	$\langle x_1, \ldots, x_n := a_1, \ldots, a_n, \sigma = (\eta, out)\rangle \longrightarrow (\eta[x_1 \mapsto \mathcal{A}[\![a_1]\!])\eta] \ldots [x_n \mapsto \mathcal{A}[\![a_n]\!])\eta], out)$
$[cont\_fire]$	$\langle \texttt{continue}\ , \sigma\rangle \longrightarrow [\sigma]^{continue}$
$[break\_fire]$	$\langle \texttt{break}\ , \sigma\rangle \longrightarrow [\sigma]^{break}$
$[cont\_prop]$	$\dfrac{\langle S_1, \sigma\rangle \longrightarrow [\sigma']^{continue}}{\langle S_1; S_2, \sigma\rangle \longrightarrow [\sigma']^{continue}}$
$[break\_prop]$	$\dfrac{\langle S_1, \sigma\rangle \longrightarrow [\sigma']^{break}}{\langle S_1; S_2, \sigma\rangle \longrightarrow [\sigma']^{break}}$
$[cont\_catch]$	$\dfrac{\langle S, \sigma\rangle \longrightarrow [\sigma']^{continue}}{\langle \texttt{while}\ [b]^\ell S, \sigma\rangle \longrightarrow \langle \texttt{while}\ [b]^\ell S, \sigma'\rangle}$
$[break\_catch]$	$\dfrac{\langle S, \sigma\rangle \longrightarrow [\sigma']^{break}}{\langle \texttt{while}\ [b]^\ell S, \sigma\rangle \longrightarrow \sigma}$