

# Program Analysis (70020)

## Data Flow Analysis

Herbert Wiklicky

Department of Computing  
Imperial College London

herbert@doc.ic.ac.uk  
h.wiklicky@imperial.ac.uk

Autumn 2022

1/51

## Data Flow Analysis

The general approach for determining program properties for procedural languages via a dataflow analysis:

- ▶ Extract Data Flow Information
- ▶ Formulate Data Flow Equations
  - ▶ Update Local Information
  - ▶ Collect Global Information
- ▶ Construct Solution(s) of Equations

2/51

# Available Expressions

The *Available Expressions Analysis* will determine:

*For each program point, which expressions **must** (are guaranteed to) have already been computed, and not later modified, on **all paths** to that program point.*

This information can be used to avoid the re-computation of an expression. For clarity, we will concentrate on arithmetic expressions.

3/51

## Example

Consider the following simple program:

```
[  $x := a + b$  ]1;  
[  $y := a * b$  ]2;  
while [  $y > a + b$  ]3 do (  
    [  $a := a + 1$  ]4;  
    [  $x := a + b$  ]5 )
```

It should be clear that the expression  $a+b$  is available every time the execution reaches the test (label 3) in the loop; as a consequence, the expression need not be recomputed.

4/51

# AE Analysis

$$kill_{AE} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$gen_{AE} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$AE_{entry} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$AE_{exit} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

5/51

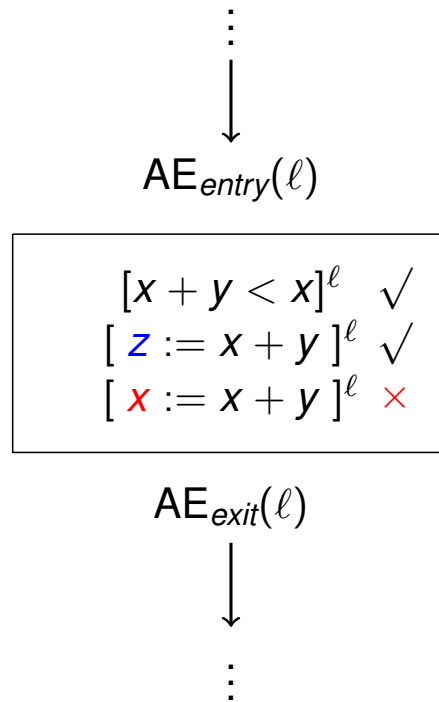
## AE Auxiliary Functions

$$\begin{aligned} kill_{AE}([x := a]^\ell) &= \{a' \in \mathbf{AExp}_* \mid x \in FV(a')\} \\ kill_{AE}([\mathbf{skip}]^\ell) &= \emptyset \\ kill_{AE}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} gen_{AE}([x := a]^\ell) &= \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\ gen_{AE}([\mathbf{skip}]^\ell) &= \emptyset \\ gen_{AE}([b]^\ell) &= \mathbf{AExp}(b) \end{aligned}$$

6/51

## AE Local Change



Whenever a variable  $x$  in an expression gets a new value the expression becomes unavailable.

7/51

## AE Equation Schemes

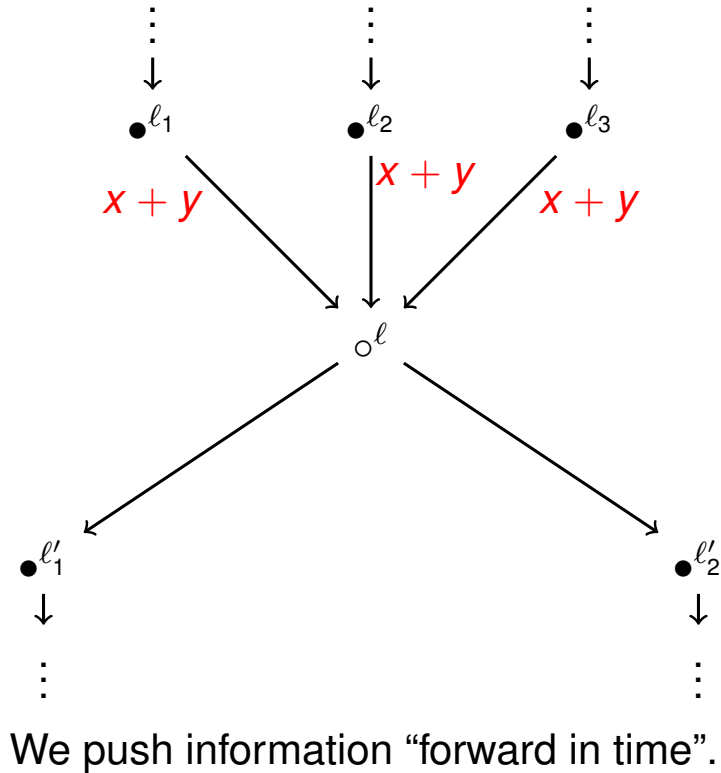
$$\text{AE}_{\text{entry}}(\ell) = \begin{cases} \emptyset, & \text{if } \ell = \text{init}(S_*) \\ \bigcap \{ \text{AE}_{\text{exit}}(\ell') \mid (\ell', \ell) \in \text{flow}(S_*) \}, & \text{otherwise} \end{cases}$$

$$\text{AE}_{\text{exit}}(\ell) = (\text{AE}_{\text{entry}}(\ell) \setminus \text{kill}_{\text{AE}}([B]^\ell)) \cup \text{gen}_{\text{AE}}([B]^\ell)$$

where  $[B]^\ell \in \text{blocks}(S_*)$

8/51

# AE Global Collection



9/51

## Largest Solution

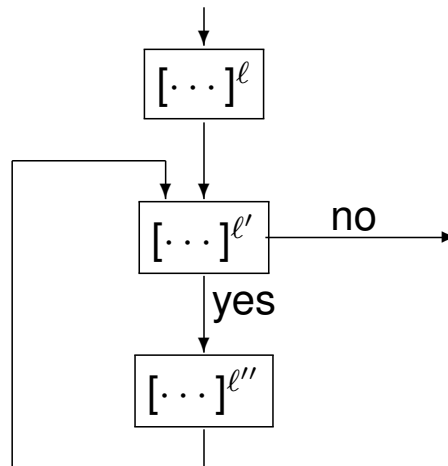
The analysis is a *forward analysis* and we are interested in the *largest* sets satisfying the equation for  $AE_{entry}$  and  $AE_{exit}$ .

$$[z := x + y]^l; \mathbf{while} [true]^{l'} \mathbf{do} [skip]^{l''}$$

$$\begin{aligned} AE_{entry}(l) &= \emptyset \\ AE_{entry}(l') &= AE_{exit}(l) \cap AE_{exit}(l'') \\ AE_{entry}(l'') &= AE_{exit}(l') \\ AE_{exit}(l) &= AE_{entry}(l) \cup \{x + y\} \\ AE_{exit}(l') &= AE_{entry}(l') \\ AE_{exit}(l'') &= AE_{entry}(l'') \end{aligned}$$

10/51

# Obtaining Solutions



After some simplification, we find that:

$$AE_{entry}(\ell') = \{x + y\} \cap AE_{entry}(\ell')$$

11/51

## AE Example

```

[ x := a + b ]1;
[ y := a * b ]2;
while [ y > a + b ]3 do (
  [ a := a + 1 ]4;
  [ x := a + b ]5 )
  
```

| $\ell$ | $kill_{AE}(\ell)$         | $gen_{AE}(\ell)$ |
|--------|---------------------------|------------------|
| 1      | $\emptyset$               | $\{a + b\}$      |
| 2      | $\emptyset$               | $\{a * b\}$      |
| 3      | $\emptyset$               | $\{a + b\}$      |
| 4      | $\{a + b, a * b, a + 1\}$ | $\emptyset$      |
| 5      | $\emptyset$               | $\{a + b\}$      |

12/51

## AE Example: Equations

```
[ x := a + b ]1;  
[ y := a * b ]2;  
while [ y > a + b ]3 do (  
    [ a := a + 1 ]4;  
    [ x := a + b ]5 )
```

$$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$$

$$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$$

$$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$$

$$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a * b, a + 1\}$$

$$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$$

13/51

## AE Example: Equations

```
[ x := a + b ]1;  
[ y := a * b ]2;  
while [ y > a + b ]3 do (  
    [ a := a + 1 ]4;  
    [ x := a + b ]5 )
```

$$AE_{entry}(1) = \emptyset$$

$$AE_{entry}(2) = AE_{exit}(1)$$

$$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$$

$$AE_{entry}(4) = AE_{exit}(3)$$

$$AE_{entry}(5) = AE_{exit}(4)$$

14/51

## AE Example: Equations

$$AE_{entry}(1) = \emptyset$$

$$AE_{entry}(2) = AE_{exit}(1)$$

$$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$$

$$AE_{entry}(4) = AE_{exit}(3)$$

$$AE_{entry}(5) = AE_{exit}(4)$$

$$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$$

$$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$$

$$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$$

$$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a * b, a + 1\}$$

$$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$$

15/51

## AE Example: Solutions

| $\ell$ | $AE_{entry}(\ell)$ | $AE_{exit}(\ell)$  |
|--------|--------------------|--------------------|
| 1      | $\emptyset$        | $\{a + b\}$        |
| 2      | $\{a + b\}$        | $\{a + b, a * b\}$ |
| 3      | $\{a + b\}$        | $\{a + b\}$        |
| 4      | $\{a + b\}$        | $\emptyset$        |
| 5      | $\emptyset$        | $\{a + b\}$        |

Note that, even though  $a$  is redefined in the loop, the expression  $a+b$  is re-evaluated in the loop and so it is always available on entry to the loop. On the other hand,  $a*b$  is available on the first entry to the loop but is killed before the next iteration.

16/51



# Reaching Definitions Analysis

The *Reaching Definitions Analysis* is analogous to the previous one except that we are interested in:

*For each program point, which assignments **may** have been made and not overwritten, when program execution reaches this point along **some path**.*

A main application of Reaching Definitions Analysis is in the construction of direct links between blocks that produce values and blocks that use them.

17/51

## Example

A simple example to illustrate the *RD* analysis would be:

```
[ x := 5 ]1;  
[ y := 1 ]2;  
while [ x > 1 ]3 do (  
    [ y := x * y ]4;  
    [ x := x - 1 ]5 )
```

All of the assignments reach the entry of 4 (the assignments labelled 1 and 2 reach there on the first iteration); only the assignments labelled 1, 4 and 5 reach the entry of 5.

18/51

# RD Analysis

$$kill_{RD} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{Var}_* \times \mathbf{Lab}_*)$$

$$gen_{RD} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{Var}_* \times \mathbf{Lab}_*)$$

$$RD_{entry} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{Var}_* \times \mathbf{Lab}_*)$$

$$RD_{exit} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{Var}_* \times \mathbf{Lab}_*)$$

**Remark:** Strictly speaking we need  $\mathcal{P}(\mathbf{Var}_* \times (\mathbf{Lab}_* \cup \{?\}))$ .

19/51

## RD Auxiliary Functions

$$\begin{aligned} kill_{RD}([x := a]^\ell) &= \{(x, ?)\} \cup \{(x, \ell') \mid \\ &\quad [B]^{\ell'} \text{ a "definition" of } x \text{ in } S_*\} \\ kill_{RD}([\mathbf{skip}]^\ell) &= \emptyset \\ kill_{RD}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} gen_{RD}([x := a]^\ell) &= \{(x, \ell)\} \\ gen_{RD}([\mathbf{skip}]^\ell) &= \emptyset \\ gen_{RD}([b]^\ell) &= \emptyset \end{aligned}$$

20/51

# RD Equation Schemes

$$RD_{entry}(\ell) = \begin{cases} \{(x, ?) \mid x \in FV(S_*)\}, & \text{if } \ell = \text{init}(S_*) \\ \bigcup \{RD_{exit}(\ell') \mid (\ell', \ell) \in \text{flow}(S_*)\}, & \text{otherwise} \end{cases}$$

$$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus \text{kill}_{RD}([B]^\ell)) \cup \text{gen}_{RD}([B]^\ell) \\ \text{where } [B]^\ell \in \text{blocks}(S_*)$$

21/51

## Smallest Solution

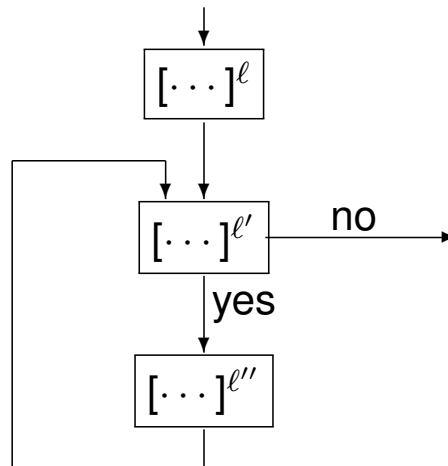
Similar to before, this is a *forward analysis* but we are interested in the *smallest* sets satisfying the equation for  $RD_{entry}$ .

$$[z := x + y]^\ell; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}$$

$$\begin{aligned} RD_{entry}(\ell) &= \{(x, ?), (y, ?), (z, ?)\} \\ RD_{entry}(\ell') &= RD_{exit}(\ell) \cup RD_{exit}(\ell'') \\ RD_{entry}(\ell'') &= RD_{exit}(\ell') \\ RD_{exit}(\ell) &= (RD_{entry}(\ell) \setminus \{(z, ?)\}) \cup \{(z, \ell)\} \\ RD_{exit}(\ell') &= RD_{entry}(\ell') \\ RD_{exit}(\ell'') &= RD_{entry}(\ell'') \end{aligned}$$

22/51

# Obtaining Solutions



After some simplification, we find that:

$$RD_{entry}(\ell') = \{(x, ?), (y, ?), (z, \ell)\} \cup RD_{entry}(\ell')$$

23/51

## RD Variations

Sometimes, when the Reaching Definitions analysis is presented in the literature, one has  $RD_{entry}(init(S_\star)) = \emptyset$  rather than  $RD_{entry}(init(S_\star)) = \{((x, ?) \mid x \in FV(S_\star))\}$ .

This is correct only for programs that always assign variables before their first use; incorrect optimisations may result if this is not the case. The advantage of our formulation is that it is always semantically sound.

24/51

## RD Example

```

[ x := 5 ]1;
[ y := 1 ]2;
while [ x > 1 ]3 do (
    [ y := x * y ]4;
    [ x := x - 1 ]5 )

```

| $\ell$ | $kill_{RD}(\ell)$            | $gen_{RD}(\ell)$ |
|--------|------------------------------|------------------|
| 1      | $\{(x, ?), (x, 1), (x, 5)\}$ | $\{(x, 1)\}$     |
| 2      | $\{(y, ?), (y, 2), (y, 4)\}$ | $\{(y, 2)\}$     |
| 3      | $\emptyset$                  | $\emptyset$      |
| 4      | $\{(y, ?), (y, 2), (y, 4)\}$ | $\{(y, 4)\}$     |
| 5      | $\{(x, ?), (x, 1), (x, 5)\}$ | $\{(x, 5)\}$     |

25/51

## RD Example: Equations

```

[ x := 5 ]1;
[ y := 1 ]2;
while [ x > 1 ]3 do (
    [ y := x * y ]4;
    [ x := x - 1 ]5 )

```

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 1)\}$$

$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 2)\}$$

$$RD_{exit}(3) = RD_{entry}(3)$$

$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 4)\}$$

$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 5)\}$$

26/51

## RD Example: Equations

```
[ x := 5 ]1;  
[ y := 1 ]2;  
while [ x > 1 ]3 do (  
    [ y := x * y ]4;  
    [ x := x - 1 ]5 )
```

$$RD_{entry}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{entry}(2) = RD_{exit}(1)$$

$$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$

$$RD_{entry}(4) = RD_{exit}(3)$$

$$RD_{entry}(5) = RD_{exit}(4)$$

27/51

## RD Example: Equations

$$RD_{entry}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{entry}(2) = RD_{exit}(1)$$

$$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$

$$RD_{entry}(4) = RD_{exit}(3)$$

$$RD_{entry}(5) = RD_{exit}(4)$$

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 1)\}$$

$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 2)\}$$

$$RD_{exit}(3) = RD_{entry}(3)$$

$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 4)\}$$

$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 5)\}$$

28/51

## RD Example: Solutions

| $\ell$ | $RD_{entry}(\ell)$                   | $RD_{exit}(\ell)$                    |
|--------|--------------------------------------|--------------------------------------|
| 1      | $\{(x, ?), (y, ?)\}$                 | $\{(y, ?), (x, 1)\}$                 |
| 2      | $\{(y, ?), (x, 1)\}$                 | $\{(x, 1), (y, 2)\}$                 |
| 3      | $\{(x, 1), (y, 2), (y, 4), (x, 5)\}$ | $\{(x, 1), (y, 2), (y, 4), (x, 5)\}$ |
| 4      | $\{(x, 1), (y, 2), (y, 4), (x, 5)\}$ | $\{(x, 1), (y, 4), (x, 5)\}$         |
| 5      | $\{(x, 1), (y, 4), (x, 5)\}$         | $\{(y, 4), (x, 5)\}$                 |

```

[ x := 5 ]1;
[ y := 1 ]2;
while [ x > 1 ]3 do (
    [ y := x * y ]4;
    [ x := x - 1 ]5 )

```

29/51

## Very Busy Expression Analysis

An expression is *very busy* at the exit from a label if, no matter what path is taken from the label, the expression *must* (is guaranteed to) always be used before any of the variables occurring in it are redefined. The aim of the *Very Busy Expressions Analysis* is to determine:

*For each program point, which expressions must (is guaranteed to) be very busy at exit from the point.*

A possible optimisation based on this information is to evaluate the expression at the block and store its value for later use; this optimisation is sometimes called *hoisting* the expression.

30/51

## Example

We illustrate this analysis with the following example:

```
if [a > b]1
  then ( [x := b - a]2;
         [y := a - b]3 )
  else ( [y := b - a]4;
         [x := a - b]5 )
```

The expressions  $a - b$  and  $b - a$  are both very busy at the start of the program (label 1). They can be hoisted resulting in a code size reduction.

31/51

## VB Analysis

$$kill_{VB} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$gen_{VB} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$VB_{entry} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

$$VB_{exit} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{AExp}_*)$$

The analysis is a *backward analysis* and we are interested in the *largest* sets satisfying the equation for  $VB_{exit}$ .

32/51



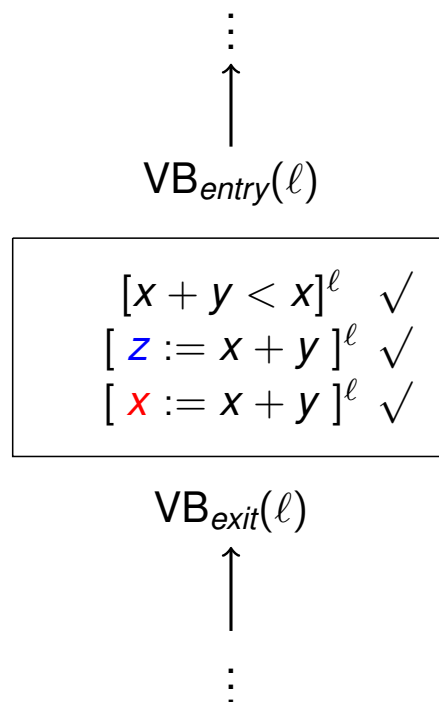
# VB Auxiliary Functions

$$\begin{aligned} kill_{VB}([x := a]^\ell) &= \{a' \in \mathbf{AExp}_* \mid x \in FV(a')\} \\ kill_{VB}([\mathbf{skip}]^\ell) &= \emptyset \\ kill_{VB}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} gen_{VB}([x := a]^\ell) &= \mathbf{AExp}(a) \\ gen_{VB}([\mathbf{skip}]^\ell) &= \emptyset \\ gen_{VB}([b]^\ell) &= \mathbf{AExp}(b) \end{aligned}$$

33/51

## VB Local Change



Whenever a variable  $x$  in an expression gets a new value it does not help us if it was evaluated before.

34/51

# VB Equation Schemes

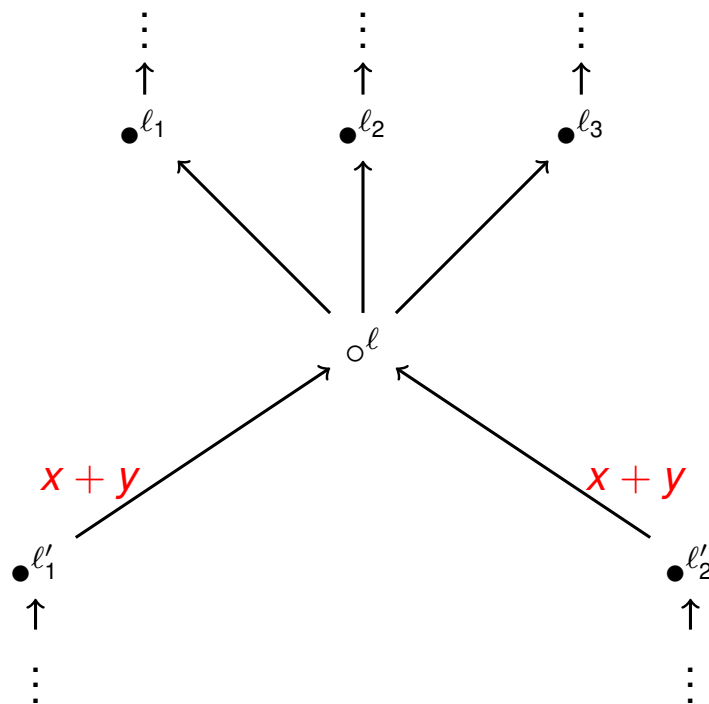
$$VB_{exit}(\ell) = \begin{cases} \emptyset, & \text{if } \ell \in final(S_*) \\ \bigcap \{VB_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_*)\}, & \text{otherwise} \end{cases}$$

$$VB_{entry}(\ell) = (VB_{exit}(\ell) \setminus kill_{VB}([B]^\ell)) \cup gen_{VB}(B^\ell)$$

where  $[B]^\ell \in blocks(S_*)$

35/51

## VB Global Collection



We need to go “back in time”.

36/51

## VB Example

```
if [a > b]1
  then ( [x := b - a]2;
         [y := a - b]3 )
  else ( [y := b - a]4;
         [x := a - b]5 )
```

| $\ell$ | $kill_{VB}(\ell)$ | $gen_{VB}(\ell)$ |
|--------|-------------------|------------------|
| 1      | $\emptyset$       | $\emptyset$      |
| 2      | $\emptyset$       | $\{b - a\}$      |
| 3      | $\emptyset$       | $\{a - b\}$      |
| 4      | $\emptyset$       | $\{b - a\}$      |
| 5      | $\emptyset$       | $\{a - b\}$      |

37/51

## VB Example: Equations

```
if [a > b]1
  then ( [x := b - a]2;
         [y := a - b]3 )
  else ( [y := b - a]4;
         [x := a - b]5 )
```

$$VB_{entry}(1) = VB_{exit}(1)$$

$$VB_{entry}(2) = VB_{exit}(2) \cup \{b - a\}$$

$$VB_{entry}(3) = \{a - b\}$$

$$VB_{entry}(4) = VB_{exit}(4) \cup \{b - a\}$$

$$VB_{entry}(5) = \{a - b\}$$

38/51

## VB Example: Equations

```
if [a > b]1
  then ( [x := b - a]2;
         [y := a - b]3 )
  else ( [y := b - a]4;
         [x := a - b]5 )
```

$$VB_{exit}(1) = VB_{entry}(2) \cap VB_{entry}(4)$$

$$VB_{exit}(2) = VB_{entry}(3)$$

$$VB_{exit}(3) = \emptyset$$

$$VB_{exit}(4) = VB_{entry}(5)$$

$$VB_{exit}(5) = \emptyset$$

39/51

## VB Example: Equations

$$VB_{entry}(1) = VB_{exit}(1)$$

$$VB_{entry}(2) = VB_{exit}(2) \cup \{b - a\}$$

$$VB_{entry}(3) = \{a - b\}$$

$$VB_{entry}(4) = VB_{exit}(4) \cup \{b - a\}$$

$$VB_{entry}(5) = \{a - b\}$$

$$VB_{exit}(1) = VB_{entry}(2) \cap VB_{entry}(4)$$

$$VB_{exit}(2) = VB_{entry}(3)$$

$$VB_{exit}(3) = \emptyset$$

$$VB_{exit}(4) = VB_{entry}(5)$$

$$VB_{exit}(5) = \emptyset$$

40/51

## VB Example: Solutions

| $\ell$ | $VB_{entry}(\ell)$ | $VB_{exit}(\ell)$  |
|--------|--------------------|--------------------|
| 1      | $\{a - b, b - a\}$ | $\{a - b, b - a\}$ |
| 2      | $\{a - b, b - a\}$ | $\{a - b\}$        |
| 3      | $\{a - b\}$        | $\emptyset$        |
| 4      | $\{a - b, b - a\}$ | $\{a - b\}$        |
| 5      | $\{a - b\}$        | $\emptyset$        |

```
if  $[a > b]^1$ 
  then (  $[x := b - a]^2;$ 
          $[y := a - b]^3$  )
  else (  $[y := b - a]^4;$ 
          $[x := a - b]^5$  )
```

41/51

## Live Variable Analysis

A variable is *live* at the exit from a label if there exists a path from the label to a use of the variable that does not re-define the variable. The *Live Variables Analysis* will determine:

*For each program point, which variables may be live at the exit from the point.*

This analysis might be used as the basis for *Dead Code Elimination*. If the variable is not live at the exit from a label then, if the elementary block is an assignment to the variable, the elementary block can be eliminated.

42/51

## Example

The example program to illustrate the *LV* analysis is:

```
[ x := 2 ]1;  
[ y := 4 ]2;  
[ x := 1 ]3;  
( if [ y > x ]4  
  then [ z := y ]5  
  else [ z := y * y ]6 );  
[ x := z ]7
```

The variable *x* is not live at the exit from 1; the first assignment to *x* is thus redundant and can be eliminated. Both *x* and *y* are alive at the exit from label 3.

43/51

## LV Analysis

$$kill_{LV} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{Var}_*)$$

$$gen_{LV} : \mathbf{Block}_* \rightarrow \mathcal{P}(\mathbf{Var}_*)$$

$$LV_{entry} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{Var}_*)$$

$$LV_{exit} : \mathbf{Lab}_* \rightarrow \mathcal{P}(\mathbf{Var}_*)$$

The analysis is a *backward analysis* and we are interested in the *smallest* sets satisfying the equation for  $LV_{exit}$ .

44/51

## LV Auxiliary Functions

$$\begin{aligned}kill_{LV}([x := a]^\ell) &= \{x\} \\kill_{LV}([\mathbf{skip}]^\ell) &= \emptyset \\kill_{LV}([b]^\ell) &= \emptyset\end{aligned}$$

$$\begin{aligned}gen_{LV}([x := a]^\ell) &= FV(a) \\gen_{LV}([\mathbf{skip}]^\ell) &= \emptyset \\gen_{LV}([b]^\ell) &= FV(b)\end{aligned}$$

45/51

## LV Equation Schemes

$$LV_{exit}(\ell) = \begin{cases} \emptyset, & \text{if } \ell \in final(S_*) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_*)\}, & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus kill_{LV}([B]^\ell) \cup gen_{LV}([B]^\ell)) \\ \text{where } [B]^\ell \in blocks(S_*)$$

46/51

## LV Example

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$   
 $(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6);$   
 $[x := z]^7$

| $\ell$ | $kill_{LV}(\ell)$ | $gen_{LV}(\ell)$ |
|--------|-------------------|------------------|
| 1      | $\{x\}$           | $\emptyset$      |
| 2      | $\{y\}$           | $\emptyset$      |
| 3      | $\{x\}$           | $\emptyset$      |
| 4      | $\emptyset$       | $\{x, y\}$       |
| 5      | $\{z\}$           | $\{y\}$          |
| 6      | $\{z\}$           | $\{y\}$          |
| 7      | $\{x\}$           | $\{z\}$          |

47/51

## LV Example: Equations

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$   
 $(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6);$   
 $[x := z]^7$

$$LV_{entry}(1) = LV_{exit}(1) \setminus \{x\}$$

$$LV_{entry}(2) = LV_{exit}(2) \setminus \{y\}$$

$$LV_{entry}(3) = LV_{exit}(3) \setminus \{x\}$$

$$LV_{entry}(4) = LV_{exit}(4) \cup \{x, y\}$$

$$LV_{entry}(5) = (LV_{exit}(5) \setminus \{z\}) \cup \{y\}$$

$$LV_{entry}(6) = (LV_{exit}(6) \setminus \{z\}) \cup \{y\}$$

$$LV_{entry}(7) = \{z\}$$

48/51



## LV Example: Equations

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$   
 $(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6);$   
 $[x := z]^7$

$$LV_{exit}(1) = LV_{entry}(2)$$

$$LV_{exit}(2) = LV_{entry}(3)$$

$$LV_{exit}(3) = LV_{entry}(4)$$

$$LV_{exit}(4) = LV_{entry}(5) \cup LV_{entry}(6)$$

$$LV_{exit}(5) = LV_{entry}(7)$$

$$LV_{exit}(6) = LV_{entry}(7)$$

$$LV_{exit}(7) = \emptyset$$

49/51

## LV Example: Solutions

| $\ell$ | $LV_{entry}(\ell)$ | $LV_{exit}(\ell)$ |
|--------|--------------------|-------------------|
| 1      | $\emptyset$        | $\emptyset$       |
| 2      | $\emptyset$        | $\{y\}$           |
| 3      | $\{y\}$            | $\{x, y\}$        |
| 4      | $\{x, y\}$         | $\{y\}$           |
| 5      | $\{y\}$            | $\{z\}$           |
| 6      | $\{y\}$            | $\{z\}$           |
| 7      | $\{z\}$            | $\emptyset$       |

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$   
 $(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6);$   
 $[x := z]^7$

50/51

## LV Variations

Some authors assume that the variables of interest are output at the end of the program.

In that case  $LV_{exit}(7)$  should be  $\{x, y, z\}$  which means that  $LV_{entry}(7)$ ,  $LV_{exit}(5)$  and  $LV_{exit}(6)$  should all be  $\{y, z\}$ .