Basic Definitions

Initial label of a statement, $init$: Stmt $\rightarrow$ Lab

$$
\begin{aligned}
init([x := a]^l) &= l \\
init([skip]^l) &= l \\
init(S_1; S_2) &= init(S_1) \\
init(\texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2) &= l \\
init(\texttt{while } [b]^l \texttt{ do } S) &= l
\end{aligned}
$$

Basic Definitions

Final label of a statement, $final$: Stmt $\rightarrow$ Lab

$$
\begin{array}{rcl}
final([x := a]^l) & = & l \\
final([skip]^l) & = & l \\
final(S_1; S_2) & = & final(S_2) \\
final(\texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2) & = & final(S_1) \cup final(S_2) \\
final(\texttt{while } [b]^l \texttt{ do } S) & = & l
\end{array}
$$

Basic Definitions

Access the set of statements or tests for a label using

$blocks$: Stmt $\rightarrow \mathcal{P}($Blocks$)$, where Blocks are the elementary labeled blocks

$$
\begin{aligned}
blocks([x := a]^l) &= \{[x := a]^l\} \\
blocks([skip]^l) &= \{[skip]^l\} \\
blocks(S_1; S_2) &= blocks(S_1) \cup blocks(S_2) \\
blocks(\texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2) &= \{[b]^l\} \cup \\
&\quad blocks(S_1) \cup blocks(S_2) \\
blocks(\texttt{while } [b]^l \texttt{ do } S) &= \{[b]^l\} \cup blocks(S)
\end{aligned}
$$

Basic Definitions

The set of labels in a program are given by $labels$: Stmt $\rightarrow \mathcal{P}($Lab$)$

$$labels(S) \quad = \quad \{l | [B]^l \in blocks(S)\}$$

Note that $init(s) \in labels(S)$ and $final(S) \subseteq labels(S)$

Basic Definitions

Control flow edges, or *flows*, are given by $flow$: Stmt $\rightarrow \mathcal{P}($Lab $\times$ Lab$)$

$$
\begin{array}{rcl}
flow([x := a]^l) & = & \emptyset \\
flow([skip]^l) & = & \emptyset \\
flow(S_1; S_2) & = & flow(S_1) \cup flow(S_2) \cup \\
 & & \{(l, init(S_2)) | l \in final(S_1)\} \\
flow(\texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2) & = & flow(S_1) \cup flow(S_2) \cup \\
 & & \{(l, init(S_1)), (l, init(S_2))\} \\
flow(\texttt{while } [b]^l \texttt{ do } S) & = & flow(S) \cup \{(l, init(S)\} \\
 & & \cup \{(l', l)) | l' \in final(S)\}
\end{array}
$$

Basic Definitions

$flow$ captures successor relationship its inverse

$flow^R$: Stmt $\rightarrow \mathcal{P}($Lab $\times$ Lab$)$ gives predecessor information

$$flow^R(S) \quad = \quad \{(l, l')|(l', l) \in flow(S)\}$$

Note that $init(s) \in labels(S)$ and $final(S) \subseteq labels(S)$

Assumptions

We make several simplifying assumptions in formulating analyses

- *isolated entries*, i.e., initial statement in program has no predecessors
- *isolated exits*, i.e., final statement in program has no successors
- *label consistent*, i.e., all blocks are uniquely labelled

If a program does not have these properties we will transform it to an equivalent one that does, e.g., by adding skip statements

How does this example look?

$$
\begin{array}{l}
\texttt{[x := a+b]}^1\texttt{;} \\
\texttt{[y := a*b]}^2\texttt{;} \\
\texttt{while [y>a+b]}^3\texttt{ do} \\
\texttt{(} \\
\quad\texttt{[a := a+1]}^4\texttt{;} \\
\quad\texttt{[x := a+b]}^5 \\
\texttt{);}
\end{array}
$$

Classic Intra-procedural Flow Analyses

We'll look at four analyses

- Available Expressons
- Reaching Definitions
- Very Busy Expressons
- Live Variables

We'll see what's similar and different and generalize to data flow frameworks

NB: We'll use a $*$ subscript to denote universes of values for the input program of interest, e.g., Blocks_* are the blocks in the input program S_*.

Notation

We'll use the following shorthands

- S_* is the top-level statement in the program
- $\mathrm{Lab}_* = labels(S_*)$
- $\mathrm{Var}_* = FV(S_*)$
- $\mathrm{Blocks}_* = blocks(S_*)$
- AExp_* the set of non-trivial sub-expressions in the program
- $\mathrm{AExp}(a)$ the set of non-trivial sub-expressions in a

Available Expressions

Available expressions analysis determines

> For each program point, which expressions *must* have already been computed, and not later modified, on *all paths* to the program point.

Usage of Available Expressions

```
[x := a+b]¹;
[y := a*b]²;
while [y>a+b]³ do
(
  [a := a+1]⁴;
  [x := a+b]⁵
);
[skip]⁶
```

$a + b$ has already been computed whenever statement 3 is reached so we don't have to recompute it

Denoting Available Expressions

We can describe the set of available expressions for the entry and exit of each program point as a set of expression drawn from the non-trivial syntactic expressions in the program AExp_*

- $\mathsf{AE}_{entry}(l) : \mathcal{P}(\mathsf{AExp}_*)$
- $\mathsf{AE}_{exit}(l) : \mathcal{P}(\mathsf{AExp}_*)$

The non-trivial expressions for our program are: $a + b$, $a * b$, and $a + 1$.

Flow Equations

An expression is *killed* in a block if any of its variables is defined in the block

$$kill_{\mathrm{AE}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{AExp}_*)$$

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

Flow Equations

An expression is *generated* in a block if it is evaluated in the block and none of the variables in the expression are subsequently defined in the block

$$gen_{\mathsf{AE}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{AExp}_*)$$

$$\begin{array}{rcl} gen_{\mathsf{AE}}([x := a]^l) & = & \{a' \in \mathsf{AExp}(a) | x \notin FV(a')\} \\ gen_{\mathsf{AE}}([skip]^l) & = & \emptyset \\ gen_{\mathsf{AE}}([b]^l) & = & \mathsf{AExp}(b) \end{array}$$

Flow Equations

The flow equations are now expressed using the two functions

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

and

$$\mathsf{AE}_{exit}(l) = (\mathsf{AE}_{entry}(l) \setminus kill_{\mathsf{AE}}(B^l)) \cup gen_{\mathsf{AE}}(B^l)$$

where $B^l \in blocks(S_*)$

Forward Analysis

Data flow information is propogated in execution order

Information at entries is calculated in terms of information at preceeding exits

The value at the entry to the program, i.e., $init(S_*)$, indicates that no expressions have been computed at that point.

Do we want the least set of AE facts at a point?

Consider the program

```
[x := a+b]^1;
while [true]^2 do
(
  [skip]^3
);
[skip]^4
```

What expressions are generated?

What expressions are killed?

Equations for Example

$$
\begin{aligned}
\mathsf{AE}_{entry}(1) &= \emptyset \\
\mathsf{AE}_{exit}(1) &= \mathsf{AE}_{entry}(1) \cup \{a+b\} \\
\mathsf{AE}_{entry}(2) &= \mathsf{AE}_{exit}(1) \cap \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{exit}(2) &= \mathsf{AE}_{entry}(2) \\
\mathsf{AE}_{entry}(3) &= \mathsf{AE}_{exit}(2) \\
\mathsf{AE}_{exit}(3) &= \mathsf{AE}_{entry}(3) \\
\mathsf{AE}_{entry}(4) &= \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{exit}(4) &= \mathsf{AE}_{entry}(4)
\end{aligned}
$$

Least or Greatest Solution

Simplification reduces this to:

$$\mathsf{AE}_{entry}(2) = \{a + b\} \cap \mathsf{AE}_{entry}(2)$$

Two possible solutions: $\emptyset$ and $\{a + b\}$

Which of these is most precise?

AE Example

```
[x := a+b]^1;
[y := a*b]^2;
while [y>a+b]^3 do
(
  [a := a+1]^4;
  [x := a+b]^5
);
[skip]^6
```

Gen/Kill Functions for Example

l	$kill_{\textsf{AE}}(l)$	$gen_{\textsf{AE}}(l)$
1	$\emptyset$	$\{a+b\}$
2	$\emptyset$	$\{a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\{a+b, a*b, a+1\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$
6	$\emptyset$	$\emptyset$

AE Solution for Example

Using chaotic iteration where we initialize values to AExp_* (rather than $\emptyset$)

l	$\mathsf{AE}_{entry}(l)$	$\mathsf{AE}_{exit}(l)$
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	$\{a+b\}$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$
6	$\emptyset$	$\emptyset$

Observations

Direction : forward

Solution : greatest

Initial Value: $AExp_*$

Initial Stmt Value: $\emptyset$

Combining Operator: $\cap$

Description: must

Paths: all

Reaching Definitions

Reaching definitions analysis determines

For each program point, which assignments *may* have been made and not overwritten along *some path* to that program point.

Usage of Reaching Definitions

$$
\begin{array}{l}
\texttt{[x := 5]}^{1}\texttt{;} \\
\texttt{[y := 1]}^{2}\texttt{;} \\
\texttt{while [x>1]}^{3}\texttt{ do} \\
\texttt{(} \\
\quad \texttt{[y := x*y]}^{4}\texttt{;} \\
\quad \texttt{[x := x-1]}^{5} \\
\texttt{);} \\
\texttt{[skip]}^{6}
\end{array}
$$

Assignments 1 and 2 reach the uses at 4.

Denoting Reaching Definitions

We can describe the set of reaching definitions for the entry and exit of each program point as a set of pairs of Var_* and $\text{Lab}_* \cup \{?\}$

- $\text{RD}_{entry}(l)$: $\mathcal{P}(\text{Var}_* \times (\text{Lab}_* \cup \{?\}))$
- $\text{RD}_{exit}(l)$: $\mathcal{P}(\text{Var}_* \times (\text{Lab}_* \cup \{?\}))$

Where pairs like $(x, ?)$ indicate that x is undefined in the program.

Flow Equations

A definition is *killed* in a block if its variables is defined in the block

$$kill_{\mathrm{RD}} : \textsf{Blocks}_* \to \mathcal{P}(\textsf{Var}_* \times (\textsf{Lab}_* \cup \{?\}))$$

$$\begin{array}{rcl} kill_{\textsf{RD}}([x := a]^l) & = & \{(x, ?)\} \cup \\ & & \{(x, l') | B^{l'} \in blocks(S_*) \wedge B = [x := a']\} \\ kill_{\textsf{RD}}([skip]^l) & = & \emptyset \\ kill_{\textsf{RD}}([b]^l) & = & \emptyset \end{array}$$

Flow Equations

A definition is *generated* in a block if the block assigns a value to a variable

$$gen_{\mathsf{RD}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{Var}_* \times (\mathsf{Lab}_* \cup \{?\}))$$

$$\begin{array}{rcl} gen_{\mathsf{RD}}([x := a]^l) & = & \{(x, l)\} \\ gen_{\mathsf{RD}}([skip]^l) & = & \emptyset \\ gen_{\mathsf{RD}}([b]^l) & = & \emptyset \end{array}$$

Flow Equations

The flow equations are now expressed using the two functions

$$\mathsf{RD}_{entry}(l) = \begin{cases} \{(x, ?) | x \in FV(S_*)\} & \text{if } l = init(S_*) \\ \bigcup\{\mathsf{RD}_{exit}(l') | (l', l) \in flow(S_*)\} & \text{otherwise} \end{cases}$$

and

$$\mathsf{RD}_{exit}(l) = (\mathsf{RD}_{entry}(l) \setminus kill_{\mathsf{RD}}(B^l)) \cup gen_{\mathsf{RD}}(B^l)$$

where $B^l \in blocks(S_*)$

Forward Analysis

Data flow information is propogated in execution order

Information at entries is calculated in terms of information at preceeding exits

Information at program entry, i.e., $init(S_*)$, represents the fact that variables are undefined when the program starts.

In a language like Java where all variables are guaranteed to have initial values one would use $\emptyset$ as the entry value.

Do we want the least set of RD facts at a point?

Consider the program

```
[z := x+y]^1;
while [true]^2 do
(
  [skip]^3
);
[skip]^4
```

What definitions are generated?

What definitions are killed?

Equations for Example

$$
\begin{aligned}
\mathsf{RD}_{entry}(1) &= \{(x,?),(y,?),(z,?)\} \\
\mathsf{RD}_{exit}(1) &= (\mathsf{RD}_{entry}(1) \setminus \{(z,?)\}) \cup \{(z,1)\} \\
\mathsf{RD}_{entry}(2) &= \mathsf{RD}_{exit}(1) \cup \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{exit}(2) &= \mathsf{RD}_{entry}(2) \\
\mathsf{RD}_{entry}(3) &= \mathsf{RD}_{exit}(2) \\
\mathsf{RD}_{exit}(3) &= \mathsf{RD}_{entry}(3) \\
\mathsf{RD}_{entry}(4) &= \mathsf{RD}_{exit}(3) \\
\mathsf{RD}_{exit}(4) &= \mathsf{RD}_{entry}(4)
\end{aligned}
$$

Least or Greatest Solution

Simplification reduces this to:

$$\mathsf{RD}_{entry}(2) = \{(x, ?), (y, ?), (z, 1)\} \cup \mathsf{RD}_{entry}(2)$$

Lots of possible solutions: any superset of $\{(x, ?), (y, ?), (z, 1)\}$

Which of these is most precise?

RD Example

```
[x := 5]1;
[y := 1]2;
while [x>1]3 do
(
   [y := x*y]4;
   [x := x-1]5
);
[skip]6
```

Gen/Kill Functions for Example

l	$kill_{\mathsf{RD}}(l)$	$gen_{\mathsf{RD}}(l)$
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	$\{(x,?),(x,1),(x,5)\}$	$\{(x,5)\}$
6	$\emptyset$	$\emptyset$

RD Solution for Example

Using chaotic iteration where we initialize values to $\emptyset$

l	$\text{RD}_{entry}(l)$	$\text{RD}_{exit}(l)$
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	$\{(x,1),(y,?)\}$	$\{(x,1),(y,2)\}$
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	$\{(x,1),(y,4),(x,5)\}$	$\{(y,4),(x,5)\}$
6	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,2),(y,4),(x,5)\}$

Observations

Direction : forward, forward

Solution : greatest, least

Initial Value: AExp$_*$, $\emptyset$

Initial Stmt Value: $\emptyset$, $\{(x, ?), (y, ?)\}$

Combining Operator: $\cap$, $\cup$

Description: must, may

Paths: all, some

Very Busy Expressions

Very busy expressions analysis determines

For each program point, which expressions *must* be used before any variable occuring in them is redefined along *all paths* leading from the exit of the program point.

Usage of Very Busy Expressions

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

$a - b$ and $b - a$ have subsequent uses on all paths leading from the conditional and are therefore *very busy*. We can hoist their calculation out of the conditional bodies.

Denoting Very Busy Expressions

We can describe the set of very busy expressions for the entry and exit of each program point as a set of expression drawn from the non-trivial syntactic expressions in the program AExp_*

- $\mathsf{VB}_{entry}(l) : \mathcal{P}(\mathsf{AExp}_*)$
- $\mathsf{VB}_{exit}(l) : \mathcal{P}(\mathsf{AExp}_*)$

The non-trivial expressions for our program are: $a - b$ and $b - a$.

Flow Equations

An expression is *killed* in a block if any of its variables is defined in the block

$$kill_{\mathrm{VB}} : \textsf{Blocks}_* \rightarrow \mathcal{P}(\textsf{AExp}_*)$$

$$\begin{array}{rcl} kill_{\textsf{VB}}([x := a]^l) & = & \{a' \in \textsf{AExp}_* | x \in FV(a')\} \\ kill_{\textsf{VB}}([skip]^l) & = & \emptyset \\ kill_{\textsf{VB}}([b]^l) & = & \emptyset \end{array}$$

Flow Equations

A very busy expression is *generated* in a block if it is evaluated in the block

$$gen_{\mathsf{VB}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{AExp}_*)$$

$$\begin{array}{rcl} gen_{\mathsf{VB}}([x := a]^l) & = & \mathsf{AExp}(a) \\ gen_{\mathsf{VB}}([skip]^l) & = & \emptyset \\ gen_{\mathsf{VB}}([b]^l) & = & \mathsf{AExp}(b) \end{array}$$

Flow Equations

The flow equations are now expressed using the two functions

$$\mathsf{VB}_{exit}(l) = \begin{cases} \emptyset & \text{if } l \in final(S_*) \\ \bigcap\{\mathsf{VB}_{entry}(l') | (l', l) \in flow^R(S_*)\} & \text{otherwise} \end{cases}$$

and

$$\mathsf{VB}_{entry}(l) = (\mathsf{VB}_{exit}(l) \setminus kill_{\mathsf{VB}}(B^l)) \cup gen_{\mathsf{VB}}(B^l)$$

where $B^l \in blocks(S_*)$

Backward Analysis

Data flow information is propogated opposite to execution order

Information at exits is calculated in terms of information at succeeding entries

Statement entry information is calculated from exit information

Information at exit from final program blocks, i.e., $final(S_*)$, represents the fact that there are no uses of expressions after termination.

Do we want the least set of VB facts at a point?

Consider the program

```
[skip]^1;
while [x>1]^2 do
(
  [skip]^3
);
[x := x+1]^4
```

What expressions are generated?

What expressions are killed?

Equations for Example

$$
\begin{aligned}
\mathsf{VB}_{entry}(1) &= \mathsf{VB}_{exit}(1) \\
\mathsf{VB}_{exit}(1) &= \mathsf{VB}_{entry}(2) \\
\mathsf{VB}_{entry}(2) &= \mathsf{VB}_{exit}(2) \\
\mathsf{VB}_{exit}(2) &= \mathsf{VB}_{entry}(3) \cap \mathsf{VB}_{entry}(4) \\
\mathsf{VB}_{entry}(3) &= \mathsf{VB}_{exit}(3) \\
\mathsf{VB}_{exit}(3) &= \mathsf{VB}_{entry}(2) \\
\mathsf{VB}_{entry}(4) &= \{x + 1\} \\
\mathsf{VB}_{exit}(4) &= \emptyset
\end{aligned}
$$

Least or Greatest Solution

Simplification reduces this to:

$$\mathsf{VB}_{entry}(2) = \{x+1\} \cap \mathsf{VB}_{entry}(2)$$

Two possible solutions: $\emptyset$ and $\{x+1\}$

Which of these is most precise?

VB Example

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

Gen/Kill Functions for Example

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

VB Solution for Example

Using chaotic iteration where we initialize values to AExp_* (rather than $\emptyset$)

l	$\text{VB}_{entry}(l)$	$\text{VB}_{exit}(l)$
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	$\{a-b\}$	$\emptyset$

Observations

Direction : forward, forward, backward

Solution : greatest, least, greatest

Initial Value: AExp$_*$, $\emptyset$, AExp$_*$

Initial Stmt Value: $\emptyset$, $\{(x, ?), (y, ?)\}$, $\emptyset$

Combining Operator: $\cap$, $\cup$, $\cap$

Description: must, may, must

Paths: all, some, all

Live Variables

Live variables analysis determines

> For each program point, which variables *must* have a subsequent use on *some path* from the program point prior to the next definition of that variable.

Usage of Live Variables

```
[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
```

Since the value assigned at 1 is never used, the assignment can be removed.

Denoting Live Variables

We can describe the set of live variables for the entry and exit of each program point as a subset of variables in the program Var_*

- $\mathsf{LV}_{entry}(l) : \mathcal{P}(\mathsf{Var}_*)$
- $\mathsf{LV}_{exit}(l) : \mathcal{P}(\mathsf{Var}_*)$

Flow Equations

A variable is *killed* in a block if it is defined in the block

$$kill_{\mathrm{LV}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{Var}_*)$$

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

Flow Equations

A variable (use) is *generated* in a block if it is evaluated in the block

$$gen_{\mathsf{LV}} : \mathsf{Blocks}_* \to \mathcal{P}(\mathsf{Var}_*)$$

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

Flow Equations

The flow equations are now expressed using the two functions

$$\mathsf{LV}_{exit}(l) = \begin{cases} \emptyset & \text{if } l \in final(S_*) \\ \bigcup\{\mathsf{LV}_{entry}(l') | (l', l) \in flow^R(S_*)\} & \text{otherwise} \end{cases}$$

and

$$\mathsf{LV}_{entry}(l) = (\mathsf{LV}_{exit}(l) \setminus kill_{\mathsf{LV}}(B^l)) \cup gen_{\mathsf{LV}}(B^l)$$

where $B^l \in blocks(S_*)$

Backward Analysis

Data flow information is propogated opposite to execution order

Information at flows in at the final node since we are interested in future uses of variables

For whole programs, there are no uses subsequent to a final node so we use $\emptyset$ as the exit value of final statements.

If we are analyzing a part of a program we would take Var_* as the initial value since we must there are subsequent uses.

Do we want the least set of LV facts at a point?

Consider the program

```
[skip]^1;
while [x>1]^2 do
(
  [skip]^3
);
[x := x+1]^4
```

What expressions are generated?

What expressions are killed?

Equations for Example

$$
\begin{aligned}
\mathsf{LV}_{entry}(1) &= \mathsf{LV}_{exit}(1) \\
\mathsf{LV}_{exit}(1) &= \mathsf{LV}_{entry}(2) \\
\mathsf{LV}_{entry}(2) &= \mathsf{LV}_{exit}(2) \cup \{x\} \\
\mathsf{LV}_{exit}(2) &= \mathsf{LV}_{entry}(3) \cup \mathsf{LV}_{entry}(4) \\
\mathsf{LV}_{entry}(3) &= \mathsf{LV}_{exit}(3) \\
\mathsf{LV}_{exit}(3) &= \mathsf{LV}_{entry}(2) \\
\mathsf{LV}_{entry}(4) &= \{x\} \\
\mathsf{LV}_{exit}(4) &= \emptyset
\end{aligned}
$$

Least or Greatest Solution

Simplification reduces this to:

$$\mathsf{LV}_{exit}(2) = \{x\} \cup \mathsf{LV}_{exit}(2)$$

Lots of possible solutions: any superset of $\{x\}$

Which of these is most precise?

LV Example

```
[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
```

Gen/Kill Functions for Example

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

LV Solution for Example

Using chaotic iteration where we initialize values to $\emptyset$

l	$\mathsf{LV}_{entry}(l)$	$\mathsf{LV}_{exit}(l)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
6	$\{z\}$	$\emptyset$

Observations

Direction : forward, forward, backward, backward

Solution : greatest, least, greatest, least

Initial Value: AExp_*, $\emptyset$, AExp_*, $\emptyset$

Initial Stmt Value: $\emptyset$, $\{(x, ?), (y, ?)\}$, $\emptyset$, $\emptyset$

Combining Operator: $\cap$, $\cup$, $\cap$, $\cup$

Description: must, may, must, may

Paths: all, some, all, some