17. Data Flow Analysis

Goal: Analyze the behavior of programs statically,
i.e., at compile-time.

Approach: Compute a fixed point on an abstract domain
of data flow values.

17.1 While-Programs

Definition:

- The syntax of labelled while-programs
is given by the following BNF:

$a ::= k \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$ // Arithmetic expressions

$b ::= true \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2$ // Boolean expressions

$c ::= [skip]^l \mid [x := a]^l \mid c_1 ; c_2$ // Programs
$\mid$ if $[b]^l$ then c_1 else c_2 fi
$\mid$ while $[b]^l$ do c od,

where $k \in \mathbb{Z}$, $true \in \mathbb{B}$, $x \in Var$, and $l \in Lab$,
with Var and Lab sets of variables and labels, respectively.
Note that each program uses finitely many variables and labels.

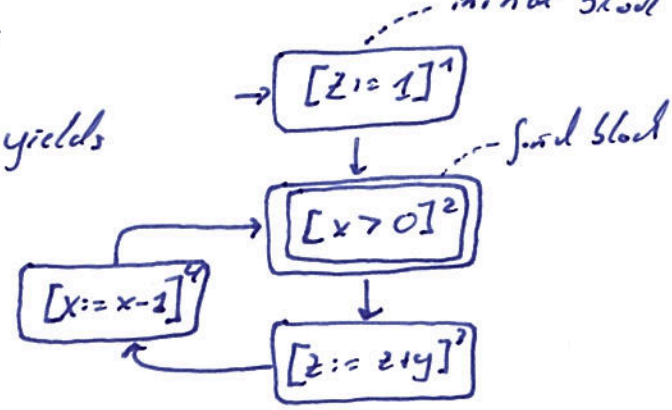
- Labelled commands are called blocks,
and we assume that blocks are uniquely labelled.

- Programs can be represented as control-flow graphs $G = (B, E, F)$
with · B the set of blocks in the program,
· $E \subseteq B$ a set of extremal blocks (initial or final), and
· $F \subseteq B \times B$ the flow relation.

- Typically, a control-flow graph represents the structure of the program of interest:

```
c = [z := 1]^1
    while [x > 0]^2 do
      [z := z * y]^3;
      [x := x - 1]^4
    od
```

yields



- For control-flow graphs, we assume that
  - the initial block has no incoming edges and
  - the final blocks have no outgoing edges.

This form can always be achieved by adding skip-commands.
The example above satisfies the condition on initial blocks but violates the condition on final blocks.

- A control-flow graph $G = (B, E, F)$ will either have the initial block as the only extremal block or the final blocks as the set of extremal blocks, but not both.
The point is that we conduct a data flow analysis either forward, from the initial block, or backward, from the final blocks.

- Indeed, there are data flow analyses that proceed against program order (backwards, e.g. live-variables). In this case, the control-flow graph does not represent the structure of the program but turns around the flow edges and starts from the final blocks.

- In general, we will always precisely define the control-flow graph of interest.

17.2 Monotone Frameworks

Monotone frameworks use a complete lattice with (ACC) as an abstract data domain, and mimic the commands of the program by monotone functions.

Definition:

A *data flow system* or *instance of a data flow analysis* is a triple $S = (G, (D, \leq), i, TF)$ with
- $G = (B, E, F)$ a *control-flow graph*,
- $(D, \leq)$ a complete lattice of *data flow values* satisfying (ACC),
- $i \in D$ an *initial value* for extremal blocks, and
- $TF = \{ f_b : D \to D \text{ monotone} \mid b \in B \}$ a family of *transfer functions*, one for each block, all monotone.

A data flow system $S = (G, (D, \leq), i, TF)$ *induces a system of equations* as follows:
- There is a variable X_b for each block $b \in B$
- For the variable, we have the equation

$$X_b = \begin{cases} i & \text{, if } b \in E \\ \bigsqcup \{ f_{b'}(X_{b'}) \mid (b', b) \in F \}, & \text{otherwise.} \end{cases}$$

A vector $(d_1, \ldots, d_{|B|}) \in D^{|B|}$ is a *solution to (the system of equations induced by) S*, if

$$d_b = \begin{cases} i & \text{, if } b \in E \\ \bigsqcup \{ f_{b'}(d_{b'}) \mid (b', b) \in F \}, & \text{otherwise.} \end{cases}$$

To relate *solutions* to the system of equations induced by S and *fixed points*, define the function:

$$g_S : D^{|B|} \to D^{|B|}$$
$$(d_1, \ldots, d_{|B|}) \mapsto (d_1', \ldots, d_{|B|}')$$

by

$$d_b' := \begin{cases} i & \text{, if } b \in E \\ \bigsqcup \{ f_{b'}(d_{b'}) \mid (b', b) \in F \}, & \text{otherwise.} \end{cases}$$

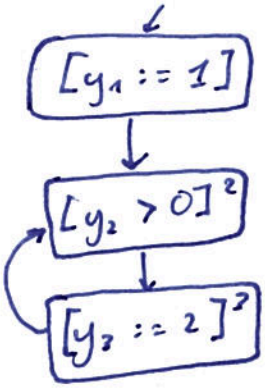
Theorem:
Vector $d \in D^{|B|}$ is a solution to the system of equations induced by S iff d is a fixed point of g_S.

We are typically interested in the least fixed point, as larger fixed points mean a loss of analysis precision.
We know how to compute least fixed points by Kleene iteration.

Example:
- We define a program analysis that computes the set of variables that could have been written when reading a block.
- Consider the program

$$c = [y_1 := 1]^1;\ \underline{\text{while}}\ [y_2 > 0]^2\ \underline{\text{do}}\ [y_3 := 2]^3\ \underline{\text{od}}$$

with $G =$



The data flow system is

$$S = (G, (\mathcal{P}(\{y_1, y_2, y_3\}), \subseteq), \emptyset, \{f_1, f_2, f_3\})$$

with $f_1, f_2, f_3 : \mathcal{P}(\{y_1, y_2, y_3\}) \to \mathcal{P}(\{y_1, y_2, y_3\})$

$$f_1(X) := X \cup \{y_1\} \qquad f_2(X) := X \qquad f_3(X) := X \cup \{y_3\}.$$

The data flow system induces the system of equations

$$X_1 = \emptyset$$

$$X_2 = \underbrace{(X_1 \cup \{y_1\})}_{= f_1(X_1)} \cup \underbrace{(X_3 \cup \{y_3\})}_{= f_3(X_3)}$$

$$X_3 = \underbrace{X_2}_{= f_2(X_2)}$$

4. A solution is $(\emptyset, \{y_1, y_3\}, \{y_1, y_3\})$. It is the least solution.

17.3 Classification of Data Flow Analyses and Examples

The explanation is given in the slides.

The programs are as follows:

Reaching Definitions:

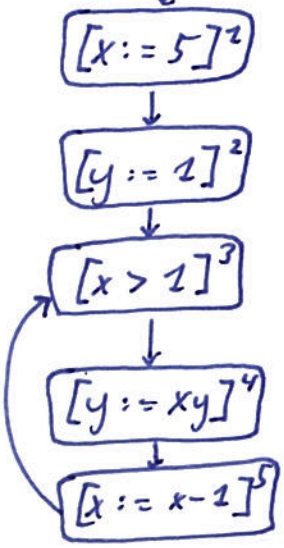
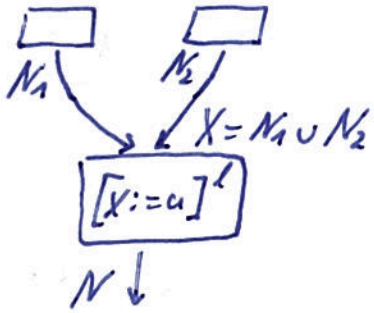


Illustration:



$= (X \setminus kill) \cup gen$.

Available Expressions:

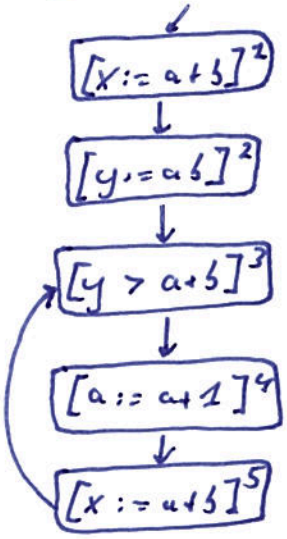
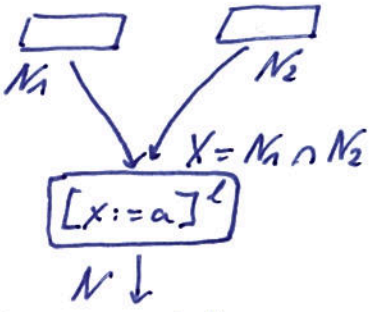


Illustration:



$= (X \setminus kill) \cup gen$

Live Variables:

```
[x := 2]^1;
[y := 4]^2;
[x := 1]^3;
if [y > 0]^4 then
  [z := x]^5
else
  [z := y]^6
fi;
[x := z]^7
```

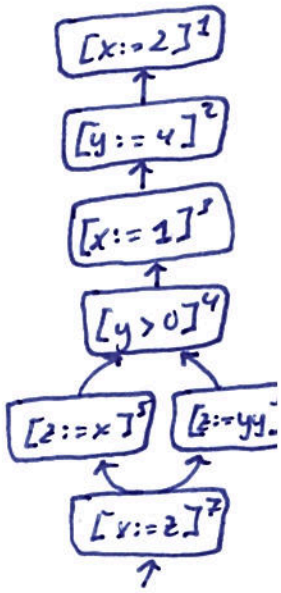
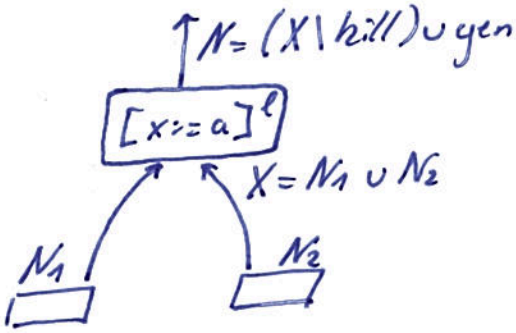


Illustration:



Busy Expressions:

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

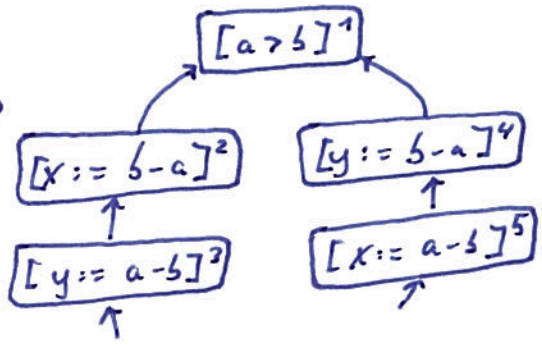


Illustration: