Program Analysis (70020)

While Language

Herbert Wiklicky

Department of Computing
Imperial College London

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

Spring 2026

Syntactic Constructs

We use the following syntactic categories:

$$
\begin{array}{lcll}
a & \in & \textbf{AExp} & \text{arithmetic expressions} \\
b & \in & \textbf{BExp} & \text{boolean expressions} \\
S & \in & \textbf{Stmt} & \text{statements}
\end{array}
$$

Abstract Syntax of WHILE

The syntax of the language WHILE is given by the following **abstract syntax**:

$$
\begin{array}{lll}
a & ::= & x \mid n \mid a_1 \; op_a \; a_2 \\
b & ::= & true \mid false \mid not \; b \mid b_1 \; op_b \; b_2 \mid a_1 \; op_r \; a_2 \\
S & ::= & x := a \\
 & & \mid \textbf{skip} \\
 & & \mid S_1 ; S_2 \\
 & & \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \\
 & & \mid \textbf{while } b \textbf{ do } S
\end{array}
$$

Syntactical Categories

We assume some countable/finite set of variables is given;

$$
\begin{array}{rcll}
x, y, z, \ldots & \in & \textbf{Var} & \text{variables} \\
n, m, \ldots & \in & \textbf{Num} & \text{numerals} \\
\ell, \ldots & \in & \textbf{Lab} & \text{labels}
\end{array}
$$

Numerals (integer constants) will not be further defined and neither will the operators:

$$
\begin{array}{rcll}
op_a & \in & \textbf{Op}_a & \text{arithmetic operators, e.g. } +, -, \times, \ldots \\
op_b & \in & \textbf{Op}_b & \text{boolean operators, e.g. } \wedge, \vee, \ldots \\
op_r & \in & \textbf{Op}_r & \text{relational operators, e.g. } =, <, \leq, \ldots
\end{array}
$$

Labelled Syntax of WHILE

The labelled syntax of the language WHILE is given by the following **abstract syntax**:

$$a \;::=\; x \mid n \mid a_1\ op_a\ a_2$$

$$b \;::=\; true \mid false \mid not\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$$

$$\begin{aligned} S \;::=\;& [x := a]^\ell \\ &\mid [\textbf{skip}]^\ell \\ &\mid S_1 ; S_2 \\ &\mid \textbf{if}\ [b]^\ell\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \\ &\mid \textbf{while}\ [b]^\ell\ \textbf{do}\ S \end{aligned}$$

An Example in WHILE

An example of a program written in this WHILE language is the following one which computes the factorial of the number stored in x and leaves the result in z:

$$\begin{aligned} &[\ y := x\ ]^1; \\ &[\ z := 1\ ]^2; \\ &\textbf{while}\ [y > 1]^3\ \textbf{do}\ (\\ &\qquad [\ z := z * y\ ]^4; \\ &\qquad [\ y := y - 1\ ]^5); \\ &[\ y := 0\ ]^6 \end{aligned}$$

Note the use of meta-symbols, brackets, to group statements.

Concrete Syntax of WHILE

To avoid using brackets (as meta-symbols) we could also use the **concrete syntax** of the language WHILE as follows:

$$
\begin{array}{lll}
a & ::= & x \mid n \mid a_1 \; op_a \; a_2 \\
b & ::= & true \mid false \mid not \; b \mid b_1 \; op_b \; b_2 \mid a_1 \; op_r \; a_2 \\
S & ::= & x := a \\
 & & \mid \textbf{skip} \\
 & & \mid S_1 ; S_2 \\
 & & \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \\
 & & \mid \textbf{while } b \textbf{ do } S \textbf{ od}
\end{array}
$$

Initial Label

When presenting examples of Data Flow Analyses we will use a number of operations on programs and labels. The first of these is

$$init : \textbf{Stmt} \to \textbf{Lab}$$

which returns the initial label of a statement:

$$
\begin{array}{rcl}
init([\, x := a \,]^\ell) & = & \ell \\
init([\, \textbf{skip} \,]^\ell) & = & \ell \\
init(S_1 ; S_2) & = & init(S_1) \\
init(\textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2) & = & \ell \\
init(\textbf{while } [b]^\ell \textbf{ do } S) & = & \ell
\end{array}
$$

Final Labels

We will also need a function which returns the set of final labels in a statement; whereas a sequence of statements has a single entry, it may have multiple exits (e.g. in the conditional):

$$\mathit{final} : \textbf{Stmt} \to \mathcal{P}(\textbf{Lab})$$

$$\begin{aligned}
\mathit{final}([\, x := a \,]^{\ell}) &= \{\ell\} \\
\mathit{final}([\, \textbf{skip} \,]^{\ell}) &= \{\ell\} \\
\mathit{final}(S_1;S_2) &= \mathit{final}(S_2) \\
\mathit{final}(\textbf{if } [b]^{\ell} \textbf{ then } S_1 \textbf{ else } S_2) &= \mathit{final}(S_1) \cup \mathit{final}(S_2) \\
\mathit{final}(\textbf{while } [b]^{\ell} \textbf{ do } S) &= \{\ell\}
\end{aligned}$$

The **while**-loop terminates immediately after the test fails.

Elementary Blocks

The building blocks of our analysis is given by **Block** is the set of statements, or elementary blocks, of the form:

- $[\, x := a \,]^{\ell}$, or
- $[\, \textbf{skip} \,]^{\ell}$, as well as
- tests of the form $[b]^{\ell}$.

Blocks

To access the statements or test associated with a label in a program we use the function

$$blocks : \textbf{Stmt} \rightarrow \mathcal{P}(\textbf{Block})$$

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

Labels

Then the set of labels occurring in a program is given by

$$labels : \textbf{Stmt} \rightarrow \mathcal{P}(\textbf{Lab})$$

where

$$labels(S) = \{\ell \mid [B]^\ell \in blocks(S)\}$$

Clearly $init(S) \in labels(S)$ and $final(S) \subseteq labels(S)$.

Flow

$$flow : \mathbf{Stmt} \rightarrow \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$

which maps statements to sets of flows:

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

An Example Flow

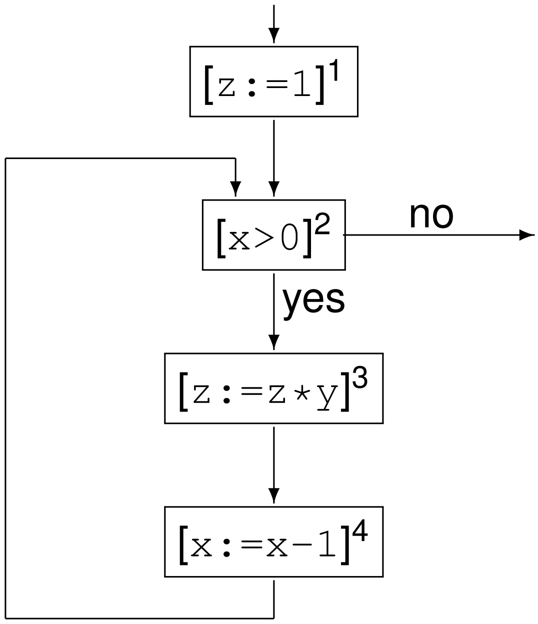
Consider the following program, power, computing the x-th power of the number stored in y:

$$\begin{array}{l}
[\,z := 1\,]^1; \\
\mathbf{while}\ [x > 1]^2\ \mathbf{do}\ (\\
\qquad [\,z := z * y\,]^3; \\
\qquad [\,x := x - 1\,]^4)
\end{array}$$

We have $labels(\text{power}) = \{1, 2, 3, 4\}$, $init(\text{power}) = 1$, and $final(\text{power}) = \{2\}$. The function *flow* produces the set:

$$flow(\text{power}) = \{(1, 2), (2, 3), (3, 4), (4, 2)\}$$

Flow Graph



Forward Analysis

The function *flow* is used in the formulation of *forward analyses*. Clearly *init*(S) is the (unique) entry node for the flow graph with nodes *labels*(S) and edges *flow*(S). Also

$$\begin{aligned} \mathit{labels}(S) &= \{\mathit{init}(S)\} \cup \\ & \quad \{\ell \mid (\ell, \ell') \in \mathit{flow}(S)\} \cup \\ & \quad \{\ell' \mid (\ell, \ell') \in \mathit{flow}(S)\} \end{aligned}$$

and for composite statements (meaning those not simply of the form $[B]^\ell$) the equation remains true when removing the $\{\mathit{init}(S)\}$ component.

Reverse Flow

In order to formulate *backward analyses* we require a function that computes reverse flows:

$$\mathit{flow}^R : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$

$$\mathit{flow}^R(S) = \{(\ell, \ell') \mid (\ell', \ell) \in \mathit{flow}(S)\}$$

For the power program, flow^R produces

$$\{(2, 1), (2, 4), (3, 2), (4, 3)\}$$

Backward Analysis

In case $\mathit{final}(S)$ contains just one element that will be the unique entry node for the flow graph with nodes $\mathit{labels}(S)$ and edges $\mathit{flow}^R(S)$. Also

$$\begin{aligned} \mathit{labels}(S) \quad = \quad & \mathit{final}(S) \ \cup \\ & \{\ell \mid (\ell, \ell') \in \mathit{flow}^R(S)\} \ \cup \\ & \{\ell' \mid (\ell, \ell') \in \mathit{flow}^R(S)\} \end{aligned}$$

Notation

We will use the notation $S_\star$ to represent the program we are analysing (the “top-level" statement) and furthermore:

- **Lab**$_\star$ to represent the labels (*labels*$(S_\star)$) appearing in $S_\star$,
- **Var**$_\star$ to represent the variables ($FV(S_\star)$) appearing in $S_\star$,
- **Block**$_\star$ to represent the elementary blocks (*blocks*$(S_\star)$) occurring in $S_\star$, and
- **AExp**$_\star$ to represent the set of *non-trivial* arithmetic subexpressions in $S_\star$ as well as
- **AExp**(a) and **AExp**(b) to refer to the set of non-trivial arithmetic subexpressions of a given arithmetic, respectively boolean, expression.

An expression is trivial if it is a single variable or constant.

Isolated Entries & Exits

Program $S_\star$ has *isolated entries* if:

$$\forall \ell \in \mathbf{Lab} : (\ell, \mathit{init}(S_\star)) \notin \mathit{flow}(S_\star)$$

This is the case whenever $S_\star$ does not start with a **while**-loop.

Similarly, we shall frequently assume that the program $S_\star$ has *isolated exits*; this means that:

$$\forall \ell_1 \in \mathit{final}(S_\star)\ \forall \ell_2 \in \mathbf{Lab} : (\ell_1, \ell_2) \notin \mathit{flow}(S_\star)$$

Label Consistency

A statement, S, is label consistent if and only if:

$$[B_1]^\ell, [B_2]^\ell \in \textit{blocks}(S) \text{ implies } B_1 = B_2$$

Clearly, if all blocks in S are uniquely labelled (meaning that each label occurs only once), then S is label consistent.

When S is label consistent the statement or clause “where $[B]^\ell \in \textit{blocks}(S)$” is unambiguous in defining a partial function from labels to elementary blocks; we shall then say that ℓ labels the block B.