Program Analysis (470)
Language Syntax

Herbert Wiklicky
herbert@doc.ic.ac.uk

Department of Computing
Imperial College London

Spring 2015

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**:

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

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

$$S \quad ::= \quad x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \mid \textbf{while } b \textbf{ do } S$$

Abstract Syntax of WHILE

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

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

Abstract Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$

$\mid$ **skip**
$\mid S_1 ; S_2$
$\mid$ **if** b **then** S_1 **else** S_2
$\mid$ **while** b **do** S

Abstract Syntax of WHILE

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

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

Abstract Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$

| **skip**
| $S_1 ; S_2$
| **if** b **then** S_1 **else** S_2
| **while** b **do** S

Abstract Syntax of WHILE

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

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

Abstract Syntax of WHILE

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

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

$$b \quad ::= \quad \mathit{true} \mid \mathit{false} \mid \mathit{not}\ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2$$

$$S \quad ::= \quad x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2 \mid \textbf{while}\ b\ \textbf{do}\ S$$

Abstract Syntax of WHILE

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

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

Abstract Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$

$\mid$ **skip**

$\mid S_1 ; S_2$

$\mid$ **if** b **then** S_1 **else** S_2

$\mid$ **while** b **do** S

Abstract Syntax of WHILE

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

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

Abstract Syntax of While

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

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

Abstract Syntax of WHILE

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

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

$$b \quad ::= \quad \mathit{true} \mid \mathit{false} \mid \mathit{not}\ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2$$

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

Abstract Syntax of WHILE

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

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

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:

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

Syntactical Categories

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

$x, y, z, \ldots$	$\in$	**Var**	variables
$n, m, \ldots$	$\in$	**Num**	numerals
$\ell, \ldots$	$\in$	**Lab**	labels

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

op_a	$\in$	$\mathbf{Op}_a$	arithmetic operators, e.g. $+, -, \times, \ldots$
op_b	$\in$	$\mathbf{Op}_b$	boolean operators, e.g. $\wedge, \vee, \ldots$
op_r	$\in$	$\mathbf{Op}_r$	relational operators, e.g. $=, <, \leq, \ldots$

Labelled Syntax of WHILE

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

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

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

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

Labelled Syntax of WHILE

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

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

Labelled Syntax of WHILE

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

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

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

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

Labelled Syntax of WHILE

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

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

Labelled Syntax of WHILE

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

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

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

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

Labelled Syntax of WHILE

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

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

Labelled Syntax of WHILE

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

$$
\begin{aligned}
a &::= x \mid n \mid a_1 \ op_a \ a_2 \\
b &::= \mathit{true} \mid \mathit{false} \mid \mathit{not}\ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2 \\
S &::= [x := a]^\ell \\
&\quad \mid [\mathbf{skip}]^\ell \\
&\quad \mid S_1 ; S_2 \\
&\quad \mid \mathbf{if}\ [b]^\ell\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \\
&\quad \mid \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S
\end{aligned}
$$

Labelled Syntax of WHILE

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

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

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

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

Labelled Syntax of WHILE

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

$$
\begin{array}{lcl}
a & ::= & x \mid n \mid a_1 \; op_a \; a_2 \\
b & ::= & \mathit{true} \mid \mathit{false} \mid \mathit{not} \; b \mid b_1 \; op_b \; b_2 \mid a_1 \; op_r \; a_2 \\
S & ::= & [x := a]^\ell \\
 & & \mid [\mathbf{skip}]^\ell \\
 & & \mid S_1 ; S_2 \\
 & & \mid \mathbf{if} \; [b]^\ell \; \mathbf{then} \; S_1 \; \mathbf{else} \; S_2 \\
 & & \mid \mathbf{while} \; [b]^\ell \; \mathbf{do} \; S
\end{array}
$$

Labelled Syntax of WHILE

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

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

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

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

Labelled Syntax of WHILE

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

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

Labelled Syntax of WHILE

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

$$
\begin{aligned}
a &::= x \mid n \mid a_1 \ op_a \ a_2 \\
b &::= true \mid false \mid not \ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2 \\
S &::= [x := a]^\ell \\
&\quad \mid [\mathbf{skip}]^\ell \\
&\quad \mid S_1 ; S_2 \\
&\quad \mid \mathbf{if} \ [b]^\ell \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \\
&\quad \mid \mathbf{while} \ [b]^\ell \ \mathbf{do} \ S
\end{aligned}
$$

Labelled Syntax of WHILE

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

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

$$b \quad ::= \quad \mathit{true} \mid \mathit{false} \mid \mathit{not}\ b \mid b_1 \ op_b \ b_2 \mid a_1 \ op_r \ a_2$$

$$S \quad ::= \quad [x := a]^\ell$$
$$\mid [\mathbf{skip}]^\ell$$
$$\mid S_1 ; S_2$$
$$\mid \mathbf{if}\ [b]^\ell\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2$$
$$\mid \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S$$

Labelled Syntax of WHILE

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

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

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:

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

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

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:

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

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

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:

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

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:

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

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

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

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$

| **skip**

| $S_1 ; S_2$

| **if** b **then** S_1 **else** S_2 **fi**

| **while** b **do** S **od**

Concrete Syntax of WHILE

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

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

Concrete Syntax of WHILE

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

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

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

$$S ::= x := a$$
$$\mid \textbf{skip}$$
$$\mid S_1 ; S_2$$
$$\mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}$$
$$\mid \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$
$$\mid \mathbf{skip}$$
$$\mid S_1 ; S_2$$
$$\mid \mathbf{if} \ b \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \ \mathbf{fi}$$
$$\mid \mathbf{while} \ b \ \mathbf{do} \ S \ \mathbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S ::= x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S ::= x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if}\; b \;\textbf{then}\; S_1 \;\textbf{else}\; S_2 \;\textbf{fi} \mid \textbf{while}\; b \;\textbf{do}\; S \;\textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if} \ b \ \textbf{then} \ S_1 \ \textbf{else} \ S_2 \ \textbf{fi} \mid \textbf{while} \ b \ \textbf{do} \ S \ \textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a \mid \textbf{skip} \mid S_1 ; S_2 \mid \textbf{if} \; b \; \textbf{then} \; S_1 \; \textbf{else} \; S_2 \; \textbf{fi} \mid \textbf{while} \; b \; \textbf{do} \; S \; \textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$
$$\mid \textbf{skip}$$
$$\mid S_1 ; S_2$$
$$\mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}$$
$$\mid \textbf{while } b \textbf{ do } S \textbf{ od}$$

Concrete Syntax of WHILE

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

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

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

$$S \quad ::= \quad x := a$$

| **skip**

| $S_1 ; S_2$

| **if** b **then** S_1 **else** S_2 **fi**

| **while** b **do** S **od**

Concrete Syntax of WHILE

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

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

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

$$S ::= x := a$$
$$\mid \textbf{skip}$$
$$\mid S_1 ; S_2$$
$$\mid \textbf{if} \; b \; \textbf{then} \; S_1 \; \textbf{else} \; S_2 \; \textbf{fi}$$
$$\mid \textbf{while} \; b \; \textbf{do} \; S \; \textbf{od}$$

Concrete Syntax of WHILE

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

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

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

$$\begin{aligned} S \quad ::= \quad & x := a \\ & \mid \textbf{skip} \\ & \mid S_1 ; S_2 \\ & \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \\ & \mid \textbf{while } b \textbf{ do } S \textbf{ od} \end{aligned}$$

Concrete Syntax of WHILE

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

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

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

$$\begin{aligned} S \quad ::= \quad & x := a \\ & \mid \textbf{skip} \\ & \mid S_1 ; S_2 \\ & \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \\ & \mid \textbf{while } b \textbf{ do } S \textbf{ od} \end{aligned}$$

Sketches of a Formal Semantics [not for exam]

Memory is modelled by an abstract state, i.e. functions of type

$$\mathbf{State} \quad = \quad \mathbf{Var} \to \mathbf{Z}.$$

For boolean and arithmetic expressions we assume that we know what they "evaluate to" in a state $s \in$ **State.** Then the semantics for **AExp** is a *total* function

$$[\![\, . \,]\!]_{\mathcal{A}} \; . \quad : \quad \mathbf{AExp} \to \mathbf{State} \to \mathbf{Z}$$

and the semantics of boolean expressions is given by

$$[\![\, . \,]\!]_{\mathcal{B}} \; . \quad : \quad \mathbf{BExp} \to \mathbf{State} \to \{\mathbf{tt}, \mathbf{ff}\}$$

Sketches of a Formal Semantics [not for exam]

Memory is modelled by an abstract state, i.e. functions of type

$$\textbf{State} \quad = \quad \textbf{Var} \to \textbf{Z}.$$

For boolean and arithmetic expressions we assume that we know what they "evaluate to" in a state $s \in \textbf{State}$. Then the semantics for **AExp** is a *total* function

$$[\![\ .\ ]\!]_{\mathcal{A}}\ . \quad : \quad \textbf{AExp} \to \textbf{State} \to \textbf{Z}$$

and the semantics of boolean expressions is given by

$$[\![\ .\ ]\!]_{\mathcal{B}}\ . \quad : \quad \textbf{BExp} \to \textbf{State} \to \{\textbf{tt}, \textbf{ff}\}$$

Sketches of a Formal Semantics [not for exam]

Memory is modelled by an abstract state, i.e. functions of type

$$\textbf{State} \quad = \quad \textbf{Var} \rightarrow \textbf{Z}.$$

For boolean and arithmetic expressions we assume that we know what they "evaluate to" in a state $s \in$ **State.** Then the semantics for **AExp** is a *total* function

$$[\![\ .\ ]\!]_{\mathcal{A}}\ .\quad : \quad \textbf{AExp} \rightarrow \textbf{State} \rightarrow \textbf{Z}$$

and the semantics of boolean expressions is given by

$$[\![\ .\ ]\!]_{\mathcal{B}}\ .\quad : \quad \textbf{BExp} \rightarrow \textbf{State} \rightarrow \{\textbf{tt}, \textbf{ff}\}$$

Evaluating Expressions [not for exam]

Let us look at a program with two variables **Var** $= \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in$ **AExp**:

$$\begin{aligned} [\![x + y]\!]_{\mathcal{A}} s_0 &= 0 + 1 = 1 \\ [\![x + y]\!]_{\mathcal{A}} s_1 &= 1 + 1 = 2 \end{aligned}$$

or a Boolean expression like $x + y \leq 1 \in$ **BExp**:

$$\begin{aligned} [\![x + y \leq 1]\!]_{\mathcal{B}} s_0 &= 1 \leq 1 = \mathbf{tt} \\ [\![x + y \leq 1]\!]_{\mathcal{B}} s_1 &= 2 \leq 1 = \mathbf{ff} \end{aligned}$$

Evaluating Expressions [not for exam]

Let us look at a program with two variables $\mathbf{Var} = \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in \mathbf{AExp}$:

$$\begin{aligned} [\![x + y]\!]_{\mathcal{A}} s_0 &= 0 + 1 = 1 \\ [\![x + y]\!]_{\mathcal{A}} s_1 &= 1 + 1 = 2 \end{aligned}$$

or a Boolean expression like $x + y \leq 1 \in \mathbf{BExp}$:

$$\begin{aligned} [\![x + y \leq 1]\!]_{\mathcal{B}} s_0 &= 1 \leq 1 = \mathbf{tt} \\ [\![x + y \leq 1]\!]_{\mathcal{B}} s_1 &= 2 \leq 1 = \mathbf{ff} \end{aligned}$$

Evaluating Expressions [not for exam]

Let us look at a program with two variables **Var** $= \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in$ **AExp**:

$$\begin{aligned} [\![x + y]\!]_{\mathcal{A}} s_0 &= 0 + 1 = 1 \\ [\![x + y]\!]_{\mathcal{A}} s_1 &= 1 + 1 = 2 \end{aligned}$$

or a Boolean expression like $x + y \leq 1 \in$ **BExp**:

$$\begin{aligned} [\![x + y \leq 1]\!]_{\mathcal{B}} s_0 &= 1 \leq 1 = \mathbf{tt} \\ [\![x + y \leq 1]\!]_{\mathcal{B}} s_1 &= 2 \leq 1 = \mathbf{ff} \end{aligned}$$

Evaluating Expressions [not for exam]

Let us look at a program with two variables **Var** $= \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in$ **AExp**:

$$\begin{aligned} [\![x + y]\!]_{\mathcal{A}} s_0 &= 0 + 1 = 1 \\ [\![x + y]\!]_{\mathcal{A}} s_1 &= 1 + 1 = 2 \end{aligned}$$

or a Boolean expression like $x + y \leq 1 \in$ **BExp**:

$$\begin{aligned} [\![x + y \leq 1]\!]_{\mathcal{B}} s_0 &= 1 \leq 1 = \mathbf{tt} \\ [\![x + y \leq 1]\!]_{\mathcal{B}} s_1 &= 2 \leq 1 = \mathbf{ff} \end{aligned}$$

Execution and Transitions [not for exam]

The configurations describe the current state of the execution.

$$\begin{array}{rcl} \langle S, s \rangle & \ldots & S \text{ is to be executed in state } s, \\ s & \ldots & \text{a terminal state (i.e. } \langle ., s \rangle). \end{array}$$

The transition relation $\Rightarrow$ specify the (possible) computational steps during the execution starting from a certain configuration

$$\langle S, s \rangle \Rightarrow \langle S', s' \rangle$$

and at the end of the computation

$$\langle S, s \rangle \Rightarrow s'$$

Execution and Transitions [not for exam]

The configurations describe the current state of the execution.

$$\begin{array}{rll} \langle S, s\rangle & \ldots & S \text{ is to be executed in state } s, \\ s & \ldots & \text{a terminal state (i.e. } \langle ., s\rangle). \end{array}$$

The transition relation $\Rightarrow$ specify the (possible) computational steps during the execution starting from a certain configuration

$$\langle S, s\rangle \Rightarrow \langle S', s'\rangle$$

and at the end of the computation

$$\langle S, s\rangle \Rightarrow s'$$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_{\mathcal{A}} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $\dfrac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_1';S_2, s'\rangle}$

(sq^T) $\dfrac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_{\mathcal{A}} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $\dfrac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$

(sq^T) $\dfrac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_{\mathcal{B}} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x:=a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_1';S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [\mathtt{x} := a]^{\ell}, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_{\mathcal{A}} s]$

(skip) $\langle [\mathbf{skip}]^{\ell}, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_1' ; S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \mathbf{tt}$

(if^F) $\langle \mathbf{if}\ [b]^{\ell}\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \mathbf{ff}$

(wh^T) $\langle \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S, s\rangle \Rightarrow \langle S; \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S, s\rangle$ if $[\![b]\!]_{\mathcal{B}} s = \mathbf{tt}$

(wh^F) $\langle \mathbf{while}\ [b]^{\ell}\ \mathbf{do}\ S, s\rangle \Rightarrow s$ if $[\![b]\!]_{\mathcal{B}} s = \mathbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_1' ; S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\textbf{skip}]^\ell, s \rangle \Rightarrow s$

(sq¹) $$\frac{\langle S_1, s \rangle \Rightarrow \langle S_1', s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_1'; S_2, s' \rangle}$$

(sq^T) $$\frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s \rangle \Rightarrow \langle S_2, s \rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s \rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s \rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s \rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [x := a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\textbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_1';S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(if^F) $\langle \textbf{if } [b]^\ell \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

(wh^T) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow \langle S; \textbf{while } [b]^\ell \textbf{ do } S, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \textbf{tt}$

(wh^F) $\langle \textbf{while } [b]^\ell \textbf{ do } S, s\rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \textbf{ff}$

Execution Rules (SOS) [not for exam]

(ass) $\langle [\mathrm{x}:=a]^\ell, s\rangle \Rightarrow s[x \mapsto [\![a]\!]_\mathcal{A} s]$

(skip) $\langle [\mathbf{skip}]^\ell, s\rangle \Rightarrow s$

(sq^1) $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_1' ; S_2, s'\rangle}$$

(sq^T) $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 ; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

(if^T) $\langle \mathbf{if}\ [b]^\ell\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \mathbf{tt}$

(if^F) $\langle \mathbf{if}\ [b]^\ell\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \mathbf{ff}$

(wh^T) $\langle \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S, s\rangle \Rightarrow \langle S; \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S, s\rangle$ if $[\![b]\!]_\mathcal{B} s = \mathbf{tt}$

(wh^F) $\langle \mathbf{while}\ [b]^\ell\ \mathbf{do}\ S, s\rangle \Rightarrow s$ if $[\![b]\!]_\mathcal{B} s = \mathbf{ff}$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

$$S \equiv [\,z := x + y\,]^{\ell}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}$$

$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0]$ and $s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lcl}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
 & \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
 & \Rightarrow & \langle \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
 & \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
 & \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

$$S \equiv [\, z := x + y \,]^{\ell}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{rcl} \langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\ & \Rightarrow & \langle [\, \textbf{skip} \,]^{\ell''}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\ & \Rightarrow & \langle \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\ & \Rightarrow & \langle [\, \textbf{skip} \,]^{\ell''}; \textbf{while } [\mathit{true}]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\ & \Rightarrow & \ldots \end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

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

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lll}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

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

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lll}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\textbf{ skip }]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\textbf{ skip }]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

$$S \equiv [\, z := x + y\,]^{\ell}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lll}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

$$S \equiv [\,z := x + y\,]^{\ell}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{rcl}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [\mathit{true}]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

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

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lcl}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\, \textbf{skip} \,]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while } [true]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\, \textbf{skip} \,]^{\ell''}; \textbf{while } [true]^{\ell'} \textbf{ do } [\, \textbf{skip} \,]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

A SOS Example [not for exam]

Consider a (perhaps rather vacuous) program like:

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

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{array}{lll}
\langle S, s_0 \rangle & \Rightarrow & \langle \textbf{while}\ [true]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [true]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle \textbf{while}\ [true]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \langle [\ \textbf{skip}\ ]^{\ell''}; \textbf{while}\ [true]^{\ell'}\ \textbf{do}\ [\ \textbf{skip}\ ]^{\ell''}, s_1 \rangle \\
& \Rightarrow & \ldots
\end{array}$$

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:

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

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):

$$\mathit{final} : \mathbf{Stmt} \rightarrow \mathcal{P}(\mathbf{Lab})$$

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

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
- [**skip** $]^\ell$, as well as
- tests of the form $[b]^\ell$.

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
- $[$ **skip** $]^\ell$, as well as
- tests of the form $[b]^\ell$.

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
- $[$ **skip** $]^{\ell}$, as well as
- tests of the form $[b]^{\ell}$.

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
- [**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

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

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

Labels

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

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

where

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

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

Flow

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

which maps statements to sets of flows:

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

An Example Flow

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

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

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

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

An Example Flow

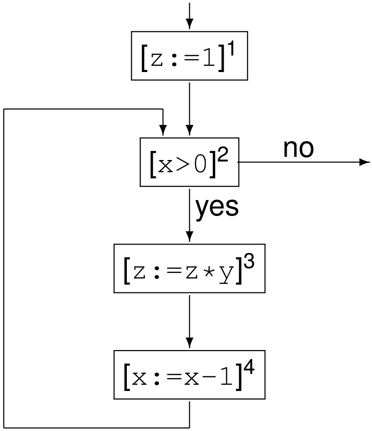
Consider the following program, power, computing the x-th power of the number stored in y:

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

We have *labels*(power) $= \{1, 2, 3, 4\}$, *init*(power) $= 1$, and *final*(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 $\mathit{init}(S)$ is the (unique) entry node for the flow graph with nodes $\mathit{labels}(S)$ and edges $\mathit{flow}(S)$. 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:

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

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

For the power program, flow^R produces

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

Backward Analysis

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

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

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.

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.

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.

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.

Notation

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

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

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.

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:

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

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:

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

Isolated Entries & Exits

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

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

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:

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

Label Consistency

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

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

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 $[B]^\ell \in \mathit{blocks}(S)$" is unambiguous in defining a partial function from labels to elementary blocks; we shall then say that ℓ labels the block B.

Label Consistency

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

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

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 $[B]^\ell \in \mathit{blocks}(S)$" is unambiguous in defining a partial function from labels to elementary blocks; we shall then say that ℓ labels the block B.