Imperative Programming Languages

Johannes Åman Pohjola
UNSW
Term 3 2022
Imperative Programming

imperō

**Definition**

*Imperative programming* is where programs are described as a series of *statements* or commands to manipulate mutable *state* or cause externally observable *effects*.

*States* may take the form of a *mapping* from variable names to their values, or even a model of a CPU state with a memory model (for example, in an *assembly language*).
The Old Days

Early microcomputer languages used a line numbering system with GO TO statements used to arrange control flow.
Factorial Example in BASIC (1964)

```
10 N = 4
20 I = 0
30 M = 1
40 IF I >= N THEN GOTO 100
50 I = I + 1
60 M = M * I
70 GOTO 40
100 PRINT M
110 END
```
The **structured programming** movement brought in *control structures* to mainstream use, such as conditionals and loops.
Factorial Example in Pascal (1970)

```
program factorial;
var n : integer;
    m : integer;
    i : integer;
begin
  n := 5;
m := 1;
i := 0;
while (i < n) do
begin
  i := i + 1;
m := m * i;
end;
println(m);
```
Syntax

We’re going to specify a language TinyImp, based on structured programming. The syntax consists of statements and expressions.

Grammar

Stmt ::= skip
  | x := Expr
  | var y · Stmt
  | if Expr then Stmt else Stmt fi
  | while Expr do Stmt od
  | Stmt ; Stmt

Expr ::= ⟨Arithmetic expressions⟩

We already know how to make unambiguous abstract syntax, so we will use concrete syntax in the rules for readability.
Examples

Example (Factorial and Fibonacci)

\[
\begin{align*}
\text{var } i & \cdot \\
\text{var } m & \cdot \\
i & := 0; \\
m & := 1; \\
\text{while } i < N \text{ do} \\
& \quad i := i + 1; \\
& \quad m := m \times i \\
\text{od}
\end{align*}
\]

\[
\begin{align*}
\text{var } m & \cdot \text{var } n & \cdot \text{var } i & \cdot \\
m & := 1; \quad n := 1; \\
i & := 1; \\
\text{while } i < N \text{ do} \\
& \quad \text{var } t \cdot t := m; \\
& \quad m := n; \\
& \quad n := m + t; \\
& \quad i := i + 1 \\
\text{od}
\end{align*}
\]
Static Semantics

Types?

- We only have one type (`int`), so type checking is a wash.
- We have to check that variables are declared before use.
- We have to check that variables are initialized before they are used!
- Set of declared free variables
- Set of initialized free variables
- Indicates that no unsafe reads occur
- Note: $V \subseteq U$
Static Semantics

**Types?** We only have one type (\texttt{int}), so type checking is a wash.

**Scopes?**
Static Semantics

Types? We only have one type (\texttt{int}), so type checking is a wash.

Scopes? We have to check that variables are declared before use.

Anything Else?
Static Semantics

Types? We only have one type (int), so type checking is a wash.

Scopes? We have to check that variables are declared before use.

Anything Else? We have to check that variables are initialized before they are used!
Static Semantics

Types? We only have one type (\(\texttt{int}\)), so type checking is a wash.

Scopes? We have to check that variables are declared before use.

Anything Else? We have to check that variables are \textit{initialized} before they are used!

\[ U; V \vdash s \texttt{ok} \leadsto W \]

Note: \( V \subseteq U \)
Static Semantics

Types? We only have one type (\texttt{int}), so type checking is a wash.

Scopes? We have to check that variables are declared before use.

Anything Else? We have to check that variables are \textit{initialized} before they are used!

\[ U; V \models s \text{ ok} \leadsto W \]

Set of declared free variables

Note: \( V \subseteq U \)
Static Semantics

**Types?** We only have one type (*int*), so type checking is a wash.

**Scopes?** We have to check that variables are declared before use.

**Anything Else?** We have to check that variables are *initialized* before they are used!

\[
U; V \vdash s \text{ ok} \leadsto W
\]

**Note:** \( V \subseteq U \)
Static Semantics

**Types?** We only have one type (`int`), so type checking is a wash.

**Scopes?** We have to check that variables are declared before use.

**Anything Else?** We have to check that variables are initialized before they are used!

\[
U; V \vdash \text{ok} \leadsto W
\]

- Set of declared free variables
- Set of initialized free variables

Indicates that no unsafe reads occur

Note: \( V \subseteq U \)
**Static Semantics**

**Types?** We only have one type (`int`), so type checking is a wash.

**Scopes?** We have to check that variables are declared before use.

**Anything Else?** We have to check that variables are *initialized* before they are used!

\[ U; V \vdash s \text{ ok} \rightsquigarrow W \]

- Set of **initialized** free variables
- Set of **declared** free variables
- Set of **definitely written to** free variables

Indicates that no unsafe reads occur

\[ V \subseteq U \]

Note:
Static Semantics Rules

\[ U; V \vdash \text{skip ok} \rightsquigarrow \emptyset \]
Static Semantics Rules

\[ U; V \vdash \text{skip ok} \Rightarrow \emptyset \]

\[ U; V \vdash x := e \text{ ok} \Rightarrow \]

\[ x \in U \text{ FV} (e) \subseteq V \]

\[ U; V \vdash x : = e \text{ ok} \Rightarrow \]

\[ U \cup \{ y \} ; V \vdash s \text{ ok} \Rightarrow W \]

\[ U \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \text{ ok} \Rightarrow \]

\[ \{ y \} \subseteq V \text{ FV} (e) \subseteq U \cup V \]

\[ U \vdash s_1 \text{ ok} \Rightarrow W \]

\[ U \vdash s_2 \text{ ok} \Rightarrow W \]

\[ U \vdash \text{while } e \text{ do } s \text{ od } \text{ ok} \Rightarrow \emptyset \]

\[ U \vdash s_1 \text{ ok} \Rightarrow W \]

\[ U \vdash s_2 \text{ ok} \Rightarrow W \]

\[ (U \cup W_1) \vdash s_2 \text{ ok} \Rightarrow W_2 \]

\[ U \vdash s_1 ; s_2 \text{ ok} \Rightarrow W \]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip ok} & \leadsto \emptyset \\
x \in U \quad & \quad U; V \vdash x := e \text{ ok} \leadsto
\end{align*}
\]
Static Semantics Rules

\[ \frac{}{U; V \vdash \text{skip ok} \rightsquigarrow \emptyset} \]

\[ \frac{x \in U \quad \text{FV}(e) \subseteq V}{U; V \vdash x := e \text{ ok} \rightsquigarrow} \]
Static Semantics Rules

\[
\begin{align*}
\text{U; V } & \vdash \text{skip ok } \leadsto \emptyset \\
\text{x } & \in \text{U} \quad \text{FV(e) } \subseteq \text{V} \\
\text{U; V } & \vdash \text{x := e ok } \leadsto \{x\}
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip} & \quad \text{ok} \leadsto \emptyset \\
x \in U & \quad \text{FV}(e) \subseteq V \\
U; V \vdash x := e \quad \text{ok} \leadsto \{x\}
\end{align*}
\]

\[
U; V \vdash \text{var} \ y \cdot s \quad \text{ok} \leadsto
\]

\[
\begin{align*}
U; V \vdash \text{if} \ e \quad \text{then} \ s_1 \quad \text{else} \ s_2 \quad \text{fi} \quad \text{ok} \leadsto \\
W_1 \cap W_2
\end{align*}
\]

\[
\begin{align*}
U; V \vdash \text{while} \ e \quad \text{do} \ s \quad \text{od} \quad \text{ok} \leadsto \\
\emptyset
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \textbf{skip} & \quad \rightarrow \emptyset \\
U; V \vdash x := e & \quad \rightarrow \{x\} \\
U \cup \{y\}; V \vdash s & \quad \rightarrow W \\
U; V \vdash \textbf{var} y \cdot s & \quad \rightarrow \\
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
& U; V \vdash \text{skip} \quad \text{ok} \quad \leadsto \emptyset \\
& U; V \vdash x := e \quad \text{ok} \quad \leadsto \{x\} \\
& U \cup \{y\}; V \vdash s \quad \text{ok} \quad \leadsto W \\
& U; V \vdash \text{var } y \cdot s \quad \text{ok} \quad \leadsto W \setminus \{y\}
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
    U; V \vdash \text{skip ok} & \leadsto \emptyset \\
    U; V \vdash x := e \text{ ok} & \leadsto \{x\} \\
    U \cup \{y\}; V \vdash s \text{ ok} & \leadsto W \\
    U; V \vdash \text{var } y \cdot s \text{ ok} & \leadsto W \setminus \{y\}
\end{align*}
\]

\[
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi ok} \leadsto
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip} \ &\text{ok} \sim \emptyset \\
x \in U &\quad \text{FV}(e) \subseteq V \\
U; V \vdash x := e \ &\text{ok} \sim \{x\} \\
U \cup \{y\}; V \vdash s \ &\text{ok} \sim W \\
U; V \vdash \text{var } y \cdot s \ &\text{ok} \sim W \setminus \{y\} \\
\text{FV}(e) \subseteq V \\
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \ &\text{fi} \ \text{ok} \sim 
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip ok} & \leadsto \emptyset \\
U; V \vdash x := e \text{ ok} & \leadsto \{x\} \\
U \cup \{y\}; V \vdash s \text{ ok} & \leadsto W \\
U; V \vdash \text{var y } \cdot \text{s ok} & \leadsto W \setminus \{y\} \\
FV(e) \subseteq V & \\
U; V \vdash s_1 \text{ ok} & \leadsto W_1 \\
U; V \vdash \text{if e then s_1 else s_2 fi ok} & \leadsto
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip} & \iff \emptyset \\
U; V \vdash x \mathbin{:=} e \text{ ok} & \iff \{x\} \\
U \cup \{y\}; V \vdash s \text{ ok} & \iff W \\
U; V \vdash \text{var } y \cdot s \text{ ok} & \iff W \setminus \{y\} \\
FV(e) \subseteq V & \\
U; V \vdash s_1 \text{ ok} & \iff W_1 \\
U; V \vdash s_2 \text{ ok} & \iff W_2 \\
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi} \text{ ok} & \iff
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V &\vdash \text{skip ok} \rightsquigarrow \emptyset & \quad & x \in U \quad \text{FV}(e) \subseteq V \\
U; V &\vdash x := e \text{ ok} \rightsquigarrow \{x\} \\
U \cup \{y\}; V &\vdash s \text{ ok} \rightsquigarrow W \\
U; V &\vdash \text{var } y \cdot s \text{ ok} \rightsquigarrow W \setminus \{y\} \\
\text{FV}(e) \subseteq V & \quad U; V \vdash s_1 \text{ ok} \rightsquigarrow W_1 & \quad U; V \vdash s_2 \text{ ok} \rightsquigarrow W_2 \\
U; V &\vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \text{ ok} \rightsquigarrow W_1 \cap W_2
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip ok} & \leadsto \emptyset \\
x \in U & \quad \text{FV}(e) \subseteq V \\
U; V \vdash x := e \text{ ok} & \leadsto \{x\} \\
U \cup \{y\}; V \vdash s \text{ ok} & \leadsto W \\
U; V \vdash \text{var } y \cdot s \text{ ok} & \leadsto W \setminus \{y\} \\
\text{FV}(e) \subseteq V & \\
U; V \vdash s_1 \text{ ok} & \leadsto W_1 \\
U; V \vdash s_2 \text{ ok} & \leadsto W_2 \\
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi ok} & \leadsto W_1 \cap W_2 \\
U; V \vdash \text{while } e \text{ do } s \text{ od ok} & \leadsto
\end{align*}
\]
**Static Semantics Rules**

\[
\begin{align*}
U; V \vdash \text{skip ok} & \leadsto \emptyset \quad & x \in U \quad \text{FV}(e) \subseteq V \\
U; V \vdash \text{x := e ok} & \leadsto \{x\} \\
U \cup \{y\}; V \vdash s \text{ ok} & \leadsto W \\
U; V \vdash \text{var y \cdot s ok} & \leadsto W \setminus \{y\} \\
\text{FV}(e) \subseteq V \quad & U; V \vdash s_1 \text{ ok} \leadsto W_1 \\
& U; V \vdash s_2 \text{ ok} \leadsto W_2 \\
& U; V \vdash \text{if e then s_1 else s_2 fi ok} \leadsto W_1 \cap W_2 \\
\text{FV}(e) \subseteq V \quad & U; V \vdash \text{while e do s od ok} \leadsto 
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip} & \quad \text{ok} \leadsto \emptyset \\
& \quad \text{if } x \in U \quad \text{FV}(e) \subseteq V \\
U; V \vdash x := e \quad \text{ok} & \quad \leadsto \{x\} \\
& \quad \text{U} \cup \{y\}; V \vdash s \quad \text{ok} \leadsto W \\
& \quad \text{U}; V \vdash \text{var } y \cdot s \quad \text{ok} \leadsto W \setminus \{y\} \\
& \quad \text{FV}(e) \subseteq V \quad \text{U}; V \vdash s_1 \quad \text{ok} \leadsto W_1 \\
& \quad \text{U}; V \vdash s_2 \quad \text{ok} \leadsto W_2 \\
& \quad \text{U}; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \quad \text{fi} \quad \text{ok} \leadsto W_1 \cap W_2 \\
& \quad \text{U}; V \vdash s \quad \text{ok} \leadsto W \\
& \quad \text{FV}(e) \subseteq V \quad \text{U}; V \vdash \text{while } e \text{ do } s \text{ od} \quad \text{ok} \leadsto \\
& \quad \text{U}; V \vdash \text{while } e \text{ do } s \text{ od} \quad \text{ok} \leadsto \\
\end{align*}
\]
## Static Semantics Rules

\[
\begin{align*}
\text{U;} \ V \vdash \text{skip ok} & \leadsto \emptyset & \text{U;} \ V \vdash x \ := \ e \ \text{ok} \leadsto \{x\} \\
\text{U} \cup \{y\}; \ V \vdash s \ \text{ok} & \leadsto W \\
\text{U} ; \ V \vdash \text{var y \cdot s ok} & \leadsto W \setminus \{y\} \\
\text{FV}(e) \subseteq V & \quad \text{U} ; \ V \vdash s_1 \ \text{ok} \leadsto W_1 \quad \text{U} ; \ V \vdash s_2 \ \text{ok} \leadsto W_2 \\
\text{U} ; \ V \vdash \text{if e then s_1 else s_2 fi ok} & \leadsto W_1 \cap W_2 \\
\text{FV}(e) \subseteq V & \quad \text{U} ; \ V \vdash s \ \text{ok} \leadsto W \\
\text{U} ; \ V \vdash \text{while e do s od ok} & \leadsto \emptyset
\end{align*}
\]
Static Semantics Rules

\[ U; V \vdash \text{skip ok} \rightsquigarrow \emptyset \]
\[ U; V \vdash x := e \text{ ok} \rightsquigarrow \{x\} \]
\[ U \cup \{y\}; V \vdash s \text{ ok} \rightsquigarrow W \]
\[ U; V \vdash \text{var y \cdot s ok} \rightsquigarrow W \setminus \{y\} \]
\[ \text{FV}(e) \subseteq V \quad U; V \vdash s_1 \text{ ok} \rightsquigarrow W_1 \quad U; V \vdash s_2 \text{ ok} \rightsquigarrow W_2 \]
\[ U; V \vdash \text{if e then s_1 else s_2 fi ok} \rightsquigarrow W_1 \cap W_2 \]
\[ \text{FV}(e) \subseteq V \quad U; V \vdash s \text{ ok} \rightsquigarrow W \]
\[ U; V \vdash \text{while e do s od ok} \rightsquigarrow \emptyset \]

\[ U; V \vdash s_1; s_2 \text{ ok} \rightsquigarrow \]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip} & \quad \Rightarrow \emptyset & x \in U \quad \text{FV}(e) \subseteq V \\
U; V \vdash x := e & \quad \Rightarrow \{x\} \\
U \cup \{y\}; V \vdash s & \quad \Rightarrow W \\
U; V \vdash \text{var } y \cdot s & \quad \Rightarrow W \setminus \{y\} \\
\text{FV}(e) \subseteq V & \\
U; V \vdash s_1 & \quad \Rightarrow W_1 \\
U; V \vdash s_2 & \quad \Rightarrow W_2 \\
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi} & \quad \Rightarrow W_1 \cap W_2 \\
\text{FV}(e) \subseteq V & \\
U; V \vdash s & \quad \Rightarrow W \\
U; V \vdash \text{while } e \text{ do } s \text{ od} & \quad \Rightarrow \emptyset \\
U; V \vdash s_1 & \quad \Rightarrow W_1 \\
U; V \vdash s_1; s_2 & \quad \Rightarrow \\
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
U; V \vdash \text{skip ok} & \rightsquigarrow \emptyset \\
U; V \vdash x := e \text{ ok} & \rightsquigarrow \{x\} \\
U \cup \{y\}; V \vdash s \text{ ok} & \rightsquigarrow W \\
U; V \vdash \text{var } y \cdot s \text{ ok} & \rightsquigarrow W \setminus \{y\} \\
FV(e) \subseteq V \quad U; V \vdash s_1 \text{ ok} & \rightsquigarrow W_1 \quad U; V \vdash s_2 \text{ ok} \rightsquigarrow W_2 \\
U; V \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi ok} & \rightsquigarrow W_1 \cap W_2 \\
FV(e) \subseteq V \quad U; V \vdash s \text{ ok} & \rightsquigarrow W \\
U; V \vdash \text{while } e \text{ do } s \text{ od ok} & \rightsquigarrow \emptyset \\
U; V \vdash s_1 \text{ ok} & \rightsquigarrow W_1 \quad U; (V \cup W_1) \vdash s_2 \text{ ok} \rightsquigarrow W_2 \\
U; V \vdash s_1; s_2 \text{ ok} & \rightsquigarrow
\end{align*}
\]
Static Semantics Rules

\[
\begin{align*}
    U; V \vdash \text{skip} & \hspace{1em} \text{ok} \leadsto \emptyset \\
    U; V \vdash x & \hspace{1em} := \ e \hspace{1em} \text{ok} \leadsto \{x\} \\
    U \cup \{y\}; V \vdash s & \hspace{1em} \text{ok} \leadsto W \\
    U; V \vdash \text{var} \ y \cdot s & \hspace{1em} \text{ok} \leadsto W \setminus \{y\}
\end{align*}
\]

\[
\begin{align*}
    \text{FV}(e) \subseteq V & \hspace{1em} U; V \vdash s_1 & \hspace{1em} \text{ok} \leadsto W_1 \\
    \text{FV}(e) \subseteq V & \hspace{1em} U; V \vdash s_2 & \hspace{1em} \text{ok} \leadsto W_2
\end{align*}
\]

\[
\begin{align*}
    U; V \vdash \text{if} \ e \hspace{1em} \text{then} \ s_1 & \hspace{1em} \text{else} \ s_2 \hspace{1em} \text{fi} \hspace{1em} \text{ok} \leadsto W_1 \cap W_2 \\
    \text{FV}(e) \subseteq V & \hspace{1em} U; V \vdash s & \hspace{1em} \text{ok} \leadsto W \\
    U; V \vdash \text{while} \ e \hspace{1em} \text{do} \ s \hspace{1em} \text{od} \hspace{1em} \text{ok} \leadsto \emptyset \\
\end{align*}
\]

\[
\begin{align*}
    U; V \vdash s_1 & \hspace{1em} \text{ok} \leadsto W_1 \\
    U; (V \cup W_1) \vdash s_2 & \hspace{1em} \text{ok} \leadsto W_2
\end{align*}
\]

\[
\begin{align*}
    U; V \vdash s_1; s_2 & \hspace{1em} \text{ok} \leadsto W_1 \cup W_2
\end{align*}
\]
Dynamic Semantics

We will use big-step operational semantics. What are the sets of evaluable expressions and values here?
Dynamic Semantics

We will use big-step operational semantics. What are the sets of evaluable expressions and values here?

Evaluable Expressions:
Dynamic Semantics

We will use **big-step operational semantics**. What are the sets of evaluable expressions and values here?

**Evaluable Expressions**: A pair containing a statement to execute and a *state* $\sigma$.

**Values**: 
Dynamic Semantics

We will use big-step operational semantics. What are the sets of evaluable expressions and values here?

**Evaluable Expressions**: A pair containing a statement to execute and a state $\sigma$.

**Values**: The final state that results from executing the statement.

**States**: mutable mappings from states to values.
States

A state is a mutable mapping from variables to their values. We use the following notation:

- To read a variable $x$ from the state $\sigma$, we write $\sigma(x)$.
States

A *state* is a mutable mapping from variables to their values. We use the following notation:

- To **read** a variable $x$ from the state $\sigma$, we write $\sigma(x)$.
- To **update** an existing variable $x$ to have value $v$ inside the state $\sigma$, we write $(\sigma : x \mapsto v)$.
States

A *state* is a mutable mapping from variables to their values. We use the following notation:

- To **read** a variable $x$ from the state $\sigma$, we write $\sigma(x)$.
- To **update** an existing variable $x$ to have value $v$ inside the state $\sigma$, we write $(\sigma : x \mapsto v)$.
- To **extend** a state $\sigma$ with a new, previously undeclared variable $x$, we write $\sigma \cdot x$. In such a state, $(\sigma \cdot x)(x)$ is undefined.
A state is a mutable mapping from variables to their values. We use the following notation:

- To read a variable $x$ from the state $\sigma$, we write $\sigma(x)$.
- To update an existing variable $x$ to have value $v$ inside the state $\sigma$, we write $\left(\sigma : x \mapsto v\right)$.
- To extend a state $\sigma$ with a new, previously undeclared variable $x$, we write $\sigma \cdot x$. In such a state, $\left(\sigma \cdot x\right)(x)$ is undefined.
- To remove a variable $x$ from the set of declared variables, we write $\left(\sigma |_x\right)$.
## States

A *state* is a mutable mapping from variables to their values. We use the following notation:

- To **read** a variable $x$ from the state $\sigma$, we write $\sigma(x)$.
- To **update** an existing variable $x$ to have value $v$ inside the state $\sigma$, we write $(\sigma : x \mapsto v)$.
- To **extend** a state $\sigma$ with a new, previously undeclared variable $x$, we write $\sigma \cdot x$. In such a state, $(\sigma \cdot x)(x)$ is undefined.
- To **remove** a variable $x$ from the set of declared variables, we write $(\sigma|_x)$.
- To exit a local scope for $x$, returning to the previous scope $\sigma'$:

$$\sigma|_{x}' = \begin{cases} 
\sigma|_x & \text{if } x \text{ is undeclared in } \sigma' \\
(\sigma|_x) \cdot x & \text{if } x \text{ is declared but undefined in } \sigma' \\
(\sigma : x \mapsto \sigma'(x)) & \text{if } \sigma'(x) \text{ is defined}
\end{cases}$$
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

$(\sigma, \text{skip}) \downarrow \sigma$
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

- $(\sigma,\text{skip}) \downarrow \sigma$
- $(\sigma_1, s_1; s_2) \downarrow$
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\frac{(\sigma, \text{skip}) \downarrow \sigma}{(\sigma, \text{skip}) \downarrow \sigma}
\frac{(\sigma_1, s_1) \downarrow \sigma_2}{(\sigma_1, s_1; s_2) \downarrow}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{array}{c}
(\sigma, \text{skip}) \Downarrow \sigma \\
(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3 \\
(\sigma_1, s_1; s_2) \Downarrow
\end{array}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{array}{c}
\frac{(\sigma, \text{skip}) \Downarrow \sigma}{(\sigma, \text{skip}) \Downarrow \sigma} & \frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma_1, s_1; s_2) \Downarrow \sigma_3}
\end{array}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\frac{(\sigma, \text{skip}) \Downarrow \sigma}{\sigma \vdash e \Downarrow v}
\]

\[
\frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma_1, s_1; s_2) \Downarrow \sigma_3}
\]

\[
\frac{\sigma \vdash e \Downarrow v}{(\sigma, x := e) \Downarrow}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) & \downarrow \sigma \\
(\sigma_1, s_1) & \downarrow \sigma_2 \\
(\sigma_2, s_2) & \downarrow \sigma_3 \\
\sigma \vdash e & \downarrow v \\
(\sigma, x := e) & \downarrow (\sigma : x \mapsto v)
\end{align*}
\]

(σ₁, s₁) ↓ σ₂ (σ₂, s₂) ↓ σ₃
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) \downarrow \sigma & \quad (\sigma_1, s_1) \downarrow \sigma_2 & (\sigma_1, s_2) \downarrow \sigma_3 \\
\sigma \vdash e \downarrow v & \quad (\sigma_1, s_1; s_2) \downarrow \sigma_3 \\
(\sigma, x := e) \downarrow (\sigma : x \mapsto v) & \quad (\sigma_1, \text{var } x \cdot s) \downarrow
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\frac{\phantom{(\sigma, skip) \Downarrow \sigma}}{(\sigma, skip) \Downarrow \sigma}
\]
\[
\frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma, \text{skip}) \Downarrow \sigma}
\]
\[
\frac{\sigma \vdash e \Downarrow v}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)}
\]
\[
\frac{(\sigma_1 \cdot x, s) \Downarrow \sigma_2}{(\sigma_1 \cdot x, s) \Downarrow \sigma_2}
\]
\[
\frac{\phantom{(\sigma_1 \cdot x, s) \Downarrow \sigma_2}}{(\sigma_1, \text{var } x \cdot s) \Downarrow}
\]
# Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

<table>
<thead>
<tr>
<th>$\sigma$, skip $\Downarrow \sigma$</th>
<th>$(\sigma_1, s_1) \Downarrow \sigma_2$</th>
<th>$(\sigma_2, s_2) \Downarrow \sigma_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma \vdash e \Downarrow v$</td>
<td>$(\sigma_1, s_1; s_2) \Downarrow \sigma_3$</td>
<td>$(\sigma_1 \cdot x, s) \Downarrow \sigma_2$</td>
</tr>
</tbody>
</table>

| $(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)$ | $(\sigma, \text{var } x \cdot s) \Downarrow \sigma_2|_{x}^{\sigma_1}$ |
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) \Downarrow \sigma \\
(\sigma, s_1 \cdot x := e) \Downarrow (\sigma : x \mapsto v) \\
(\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow
\end{align*}
\]
**Evaluation Rules**

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
\frac{}{(\sigma, \text{skip}) \Downarrow \sigma} & \quad \frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)} \quad \frac{(\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow} \\
\sigma \vdash e \Downarrow v & \quad \frac{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow}{(\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2 |_{\sigma_1}^x} \\
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) & \Downarrow \sigma & (\sigma_1, s_1) & \Downarrow \sigma_2 & (\sigma_2, s_2) & \Downarrow \sigma_3 \\
\sigma & \vdash e \Downarrow v & (\sigma_1, s_1; s_2) & \Downarrow \sigma_3 \\
(\sigma, x := e) & \Downarrow (\sigma : x \mapsto v) & (\sigma_1 \cdot x, s) & \Downarrow \sigma_2 \\
\sigma_1 & \vdash e \Downarrow v & v \neq 0 & (\sigma_1, s_1) & \Downarrow \sigma_2 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) & \Downarrow \\
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
  & (\sigma, \text{skip}) \Downarrow \sigma \\
\to & (\sigma, x := e) \Downarrow (\sigma : x \mapsto v) \\
\to & (\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \\
  & (\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2|^{\sigma_1}_{x} \\
  & (\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) \Downarrow \sigma \\
(\sigma_1, s_1) \Downarrow \sigma_2 & \quad (\sigma_2, s_2) \Downarrow \sigma_3 \\
\sigma \vdash e \Downarrow v \\
(\sigma, x := e) \Downarrow (\sigma : x \mapsto v) \\
\sigma_1 \vdash e \Downarrow v & \quad v \neq 0 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2 \\
\sigma_1 \vdash e \Downarrow 0 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
\frac{(\sigma, \text{skip}) \Downarrow \sigma}{\sigma \vdash e \Downarrow v} & \quad \frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)} \quad \frac{(\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2|_{x}^{\sigma_1}}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2} \\
\frac{\sigma_1 \vdash e \Downarrow v \quad v \neq 0}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2} & \quad \frac{\sigma_1 \vdash e \Downarrow 0 \quad (\sigma_1, s_2) \Downarrow \sigma_2}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2}
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) \Downarrow \sigma \\
(\sigma, x := e) \Downarrow (\sigma : x \mapsto v) \\
(\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2 \\
(\sigma, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation \( \sigma \vdash e \Downarrow v \) for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
\frac{\sigma, \text{skip} \Downarrow \sigma}{\sigma \vdash e \Downarrow v} \quad \frac{\sigma_1, s_1 \Downarrow \sigma_2}{(\sigma_1, s_1; s_2) \Downarrow \sigma_3} \quad \frac{(\sigma_1, x, s) \Downarrow \sigma_2}{(\sigma_1 \cdot x, s) \Downarrow \sigma_2 |_{\sigma_1}^{x}} \\
\frac{\sigma \vdash e \Downarrow v}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)} \quad \frac{\sigma_1 \vdash e \Downarrow v \quad v \neq 0}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2} \\
\frac{\sigma_1 \vdash e \Downarrow 0}{(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow} \quad \frac{\sigma_1 \vdash e \Downarrow 0}{(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow}
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
\frac{}{(\sigma, \text{skip}) \downarrow \sigma} & \quad \frac{(\sigma_1, s_1) \downarrow \sigma_2 \quad (\sigma_2, s_2) \downarrow \sigma_3}{(\sigma, x := e) \downarrow (\sigma : x \mapsto v)} & \quad \frac{(\sigma_1, s_1; s_2) \downarrow \sigma_3}{(\sigma_1 \cdot x, s) \downarrow \sigma_2} & \quad \frac{(\sigma_1 \cdot x, s) \downarrow \sigma_2}{(\sigma_1, \text{var } x \cdot s) \downarrow \sigma_2|_{x}^{\sigma_1}} \\
\frac{\sigma_1 \vdash e \downarrow v \quad v \neq 0}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \downarrow \sigma_2} & \quad \frac{\sigma_1 \vdash e \downarrow 0}{(\sigma_1, \text{while } e \text{ do } s \text{ od}) \downarrow \sigma_1} & \quad \frac{(\sigma_1, s_1) \downarrow \sigma_2}{(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \downarrow \sigma_2} & \quad \frac{(\sigma_1, s_2) \downarrow \sigma_2}{(\sigma_1, \text{while } e \text{ do } s \text{ od}) \downarrow \sigma_1}
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
&\sigma, \text{skip} \Downarrow \sigma \\
&(\sigma, x := e) \Downarrow (\sigma : x \mapsto v) \\
&(\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2
\end{align*}
\]

\[
\begin{align*}
&\sigma_1 \vdash e \Downarrow v \\
&\sigma_1 \vdash e \Downarrow 0
\end{align*}
\]

\[
\begin{align*}
&(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1
\end{align*}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{array}{c}
(\sigma, \text{skip}) \Downarrow \sigma \\
(\sigma, x := e) \Downarrow (\sigma : x \mapsto v) \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2 \\
(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1
\end{array}
\]
Evaluation Rules

We will assume we have defined a relation $\sigma \vdash e \Downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) \Downarrow \sigma & \quad \quad \quad (\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3 \\
(\sigma, x := e) \Downarrow (\sigma : x \mapsto v) & \quad \quad \quad (\sigma_1 \cdot x, s) \Downarrow \sigma_2 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) \Downarrow \sigma_2 & \quad \quad \quad (\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2|_{x_1} \\
(\sigma_1, \text{if } e \Downarrow 0 \quad (\sigma_1, s_1) \Downarrow \sigma_2 \\
(\sigma_1, \text{if } e \Downarrow 0 \quad (\sigma_1, s_2) \Downarrow \sigma_2 \\
(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1 & \quad \quad \quad (\sigma_1, s) \Downarrow \sigma_2 \quad (\sigma_2, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_3 \\
(\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \sigma_1 & \quad \quad \quad (\sigma_1, \text{while } e \text{ do } s \text{ od}) \Downarrow \\
\end{align*}
\]
**Evaluation Rules**

We will assume we have defined a relation $\sigma \vdash e \downarrow v$ for arithmetic expressions, much like in the previous lecture.

\[
\begin{align*}
(\sigma, \text{skip}) & \downarrow \sigma & (\sigma_1, s_1) & \downarrow \sigma_2 & (\sigma_2, s_2) & \downarrow \sigma_3 \\
(\sigma, x := e) & \downarrow (\sigma : x \mapsto v) & (\sigma_1, s_1; s_2) & \downarrow \sigma_3 \\
\sigma_1 & \vdash e \downarrow v & \sigma_1 & \vdash e \downarrow 0 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) & \downarrow \sigma_2 & (\sigma_1, s_1) & \downarrow \sigma_2 \\
(\sigma_1, \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi}) & \downarrow \sigma_2 & (\sigma_1, s) & \downarrow \sigma_2 \\
(\sigma_1, \text{while } e \text{ do } s \text{ od}) & \downarrow \sigma_1 & (\sigma_1, \text{while } e \text{ do } s \text{ od}) & \downarrow \sigma_3 \\
\sigma_1 & \vdash e \downarrow 0 & \sigma_1 & \vdash e \downarrow v & \sigma_1 & \vdash e \downarrow v
\end{align*}
\]
Alternative declaration semantics

What should happen when an uninitialised variable is used?

\[(\sigma \cdot y, \text{var } x \cdot y := x + 1) \Downarrow ??\]
Alternative declaration semantics

What should happen when an uninitialised variable is used?

\[(\sigma \cdot y, \text{var } x \cdot y := x + 1) \Downarrow \]

We can’t apply the assignment rule here, because in the state \(\sigma \cdot y \cdot x\), \(\sigma(x)\) is undefined.
Alternative declaration semantics

Crash and burn: \((\sigma \cdot y, \text{var } x \cdot y := x + 1) \Downarrow\)
Alternative declaration semantics

Crash and burn: \((\sigma \cdot y, \text{var } x \cdot y := x + 1) \not\Downarrow\)

\[ (\sigma_1 \cdot x, s) \Downarrow \sigma_2 \]
\[ (\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2|_{\sigma_1} \]

Default value: \((\sigma \cdot y, \text{var } x \cdot y := x + 1) \Downarrow (\sigma \cdot y) : y \mapsto 1\)

\[ ((\sigma_1 \cdot x) : x \mapsto 0, s) \Downarrow \sigma_2 \]
\[ (\sigma_1, \text{var } x \cdot s) \Downarrow \sigma_2|_{\sigma_1} \]
Alternative declaration semantics

Crash and burn: \((σ \cdot y, \text{var } x \cdot y := x + 1) \not\downarrow\)

\[
\frac{(\sigma_1 \cdot x, s) \downarrow \sigma_2}{\frac{\sigma_1, \text{var } x \cdot s \downarrow \sigma_2|_{\sigma_1}^x}{\sigma_1}}
\]

Default value: \((σ \cdot y, \text{var } x \cdot y := x + 1) \downarrow (σ \cdot y) : y \mapsto 1\)

\[
\frac{((\sigma_1 \cdot x) : x \mapsto 0, s) \downarrow \sigma_2}{\frac{\sigma_1, \text{var } x \cdot s \downarrow \sigma_2|_{\sigma_1}^x}{\sigma_1}}
\]

Junk data: \((σ \cdot y, \text{var } x \cdot y := x + 1) \downarrow (σ \cdot y) : y \mapsto 3 \text{ (or 4, or whatever we want...) \}

\[
\frac{((\sigma_1 \cdot x) : x \mapsto n, s) \downarrow \sigma_2}{\frac{\sigma_1, \text{var } x \cdot s \downarrow \sigma_2|_{\sigma_1}^x}{\sigma_1}}
\]
For a taste of axiomatic semantics, let’s define a Hoare Logic for TinyImp (without var). We write a Hoare triple judgement as:

\[
\{ \varphi \} \ s \ \{ \psi \} 
\]

Where \( \varphi \) and \( \psi \) are logical formulae about states, called assertions, and \( s \) is a statement. This triple states that if the statement \( s \) successfully evaluates from a starting state satisfying the precondition \( \varphi \), then the final state will satisfy the postcondition \( \psi \):

\[
\varphi(\sigma) \land (\sigma, s) \Downarrow \sigma' \Rightarrow \psi(\sigma')
\]
Proving Hoare Triples

To prove a Hoare triple like:

\[
\{ \text{True} \} \\
i := 0; \\
m := 1; \\
\textbf{while } i \neq N \textbf{ do} \\
\quad i := i + 1; \\
\quad m := m \times i \\
\textbf{od} \\
\{ m = N! \}
\]

We could prove this using the operational semantics. This is cumbersome, and requires an induction to deal with the \texttt{while} loop. Instead, we’ll define a set of rules to prove Hoare triples directly (called a \textit{proof calculus}).
Hoare Rules

\[
\begin{align*}
\text{(σ, skip)} & \Downarrow σ \\
\text{(σ₁, s₁)} & \Downarrow σ₂ & \text{(σ₂, s₂)} & \Downarrow σ₃ \\
\text{(σ₁, s₁; s₂)} & \Downarrow σ₃ \\
\sigma \vdash e & \Downarrow ν \\
\text{(σ, x := e)} & \Downarrow (σ : x \mapsto ν)
\end{align*}
\]
**Hoare Rules**

\[(\sigma, \text{skip}) \Downarrow \sigma\]

\[\{\varphi\} \text{ skip } \{\varphi\}\]

\[(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3\]

\[\frac{}{(\sigma_1, s_1; s_2) \Downarrow \sigma_3}\]

\[\sigma \vdash e \Downarrow v\]

\[\frac{}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)}\]

Continuing on, we can get rules for if, and while with a loop invariant:

\[
\{\varphi \land e\} s_1 \{\psi\} \quad \{\varphi \land \neg e\} s_2 \{\psi\} \quad \{\psi\}
\]

if e then s_1 else s_2 fi

\[
\{\varphi \land e\} s \{\psi\} \quad \{\varphi\}
\]

while e do s od

\[
\{\varphi \land \neg e\}
\]
Hoare Rules

\[(\sigma, \text{skip}) \Downarrow \sigma\]

\[\sigma_1 \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3 \quad (\sigma_1, s_1; s_2) \Downarrow \sigma_3\]

\[\{\varphi\} \text{ skip } \{\varphi\}\]

\[\sigma \vdash e \Downarrow v\]

\[(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)\]

Continuing on, we can get rules for if, and while with a loop invariant:
Hoare Rules

\[
\begin{align*}
(\sigma, \text{skip}) &\Downarrow \sigma \\
(\sigma_1, s_1) &\Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3 \\
(\sigma_1, s_1; s_2) &\Downarrow \sigma_3 \\
\sigma \vdash e &\Downarrow \nu \\
(\sigma, x := e) &\Downarrow (\sigma : x \mapsto \nu)
\end{align*}
\]

\[
\begin{align*}
\{\varphi\} &\text{ skip } \{\varphi\} \\
\{\varphi\} &\ s_1 \ \{\alpha\} \quad \{\alpha\} \ s_2 \ \{\psi\} \\
\{\varphi\} &\ s_1 ; \ s_2 \ \{\psi\}
\end{align*}
\]
**Hoare Rules**

\[(\sigma, \text{skip}) \Downarrow \sigma\]

\[\sigma_1, s_1 \Downarrow \sigma_2 \quad \sigma_2, s_2 \Downarrow \sigma_3 \quad \therefore (\sigma_1, s_1; s_2) \Downarrow \sigma_3\]

\[\sigma \vdash e \Downarrow v\]

\[\sigma, x := e \Downarrow (\sigma : x \mapsto v)\]

\[\{\varphi\} \text{ skip } \{\varphi\}\]

\[\varphi \quad \{\alpha\} \quad \{\alpha\} \quad s_2 \quad \psi \quad \{\varphi\} \quad s_1; s_2 \quad \psi \quad \{\varphi\}\]

\[x := e \{\varphi\}\]
Hoare Rules

\[
\begin{align*}
(\sigma, \text{skip}) & \downarrow \sigma \\
(\sigma_1, s_1) & \downarrow \sigma_2 \\
(\sigma_2, s_2) & \downarrow \sigma_3 \\
(\sigma_1, s_1; s_2) & \downarrow \sigma_3 \\
\varphi \vdash e & \downarrow v \\
\varphi[x := e] & \vdash x := e \varphi
\end{align*}
\]

Continuing on, we can get rules for if, and while with a loop invariant:

\[
\begin{align*}
\{\varphi\} \text{ if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{\psi\} \\
\{\varphi\} \text{ while } e \text{ do } s \text{ od}
\end{align*}
\]
Hoare Rules

\[\begin{align*}
&\frac{(\sigma, \text{skip}) \Downarrow \sigma}{\{\varphi\} \text{ skip} \{\varphi\}} \\
&\frac{(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3}{(\sigma_1, s_1; s_2) \Downarrow \sigma_3} \\
&\frac{\sigma \vdash e \Downarrow v}{(\sigma, x := e) \Downarrow (\sigma : x \mapsto v)} \\
&\frac{\{\varphi \land e\} \ s_1 \ \{\psi\}}{\{\varphi\} \ \text{if e then } s_1 \ \text{else } s_2 \ \text{fi} \ \{\psi\}} \\
&\frac{\{\varphi\} \ \text{while } e \ \text{do } s \ \text{od}}{\{\varphi[x := e]\} \ x := e \ \{\varphi\}}
\end{align*}\]

Continuing on, we can get rules for if, and while with a loop invariant:
Hoare Rules

\[(\sigma, \text{skip}) \downarrow \sigma\]

\[(\sigma_1, s_1) \downarrow \sigma_2 \quad (\sigma_2, s_2) \downarrow \sigma_3 \quad (\sigma_1, s_1 ; s_2) \downarrow \sigma_3\]

\[\sigma \vdash e \downarrow v\]

\[\sigma, x := e \downarrow (\sigma : x \mapsto v)\]

Continuing on, we can get rules for if, and while with a loop invariant:

\[\{\varphi \land e\} \ s_1 \ \{\psi\} \quad \{\varphi \land \neg e\} \ s_2 \ \{\psi\}\]

\[\{\varphi\} \ \text{if} \ e \ \text{then} \ s_1 \ \text{else} \ s_2 \ \text{fi} \ \{\psi\}\]

\[\{\varphi\} \ \text{while} \ e \ \text{do} \ s \ \text{od}\]
Hoare Logic

Continuing on, we can get rules for if, and while with a loop invariant:

\[
\begin{align*}
\{\varphi \land e\} s_1 \{\psi\} & \quad \{\varphi \land \neg e\} s_2 \{\psi\} \\
\{\varphi\} \text{ if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{\psi\} & \quad \{\varphi\} \text{ while } e \text{ do } s \od \{\varphi\}
\end{align*}
\]
Hoare Rules

\[(\sigma, \text{skip}) \Downarrow \sigma\]
\[\{\varphi\} \text{ skip } \{\varphi\}\]
\[(\sigma_1, s_1) \Downarrow \sigma_2 \quad (\sigma_2, s_2) \Downarrow \sigma_3\]
\[\{\varphi\} s_1 \{\alpha\} \quad \{\alpha\} s_2 \{\psi\}\]
\[(\sigma_1, s_1; s_2) \Downarrow \sigma_3\]
\[\{\varphi\} s_1; s_2 \{\psi\}\]

\[\sigma \vdash e \Downarrow v\]
\[\{\varphi[x := e]\} x := e \{\varphi\}\]

Continuing on, we can get rules for if, and while with a loop invariant:

\[\{\varphi \land e\} s_1 \{\psi\} \quad \{\varphi \land \neg e\} s_2 \{\psi\}\]
\[\{\varphi\} \text{ if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{\psi\}\]
\[\{\varphi \land e\} s \{\varphi\}\]
\[\{\varphi\} \text{ while } e \text{ do } s \text{ od } \{\varphi \land \neg e\}\]
There is one more rule, called the *rule of consequence*, that we need to insert ordinary logical reasoning into our Hoare logic proofs:

\[
\varphi \Rightarrow \alpha \quad \{ \alpha \} \ s \ \{ \beta \} \quad \beta \Rightarrow \psi \\
\{ \varphi \} \ s \ \{ \psi \}
\]
Consequence

There is one more rule, called the *rule of consequence*, that we need to insert ordinary logical reasoning into our Hoare logic proofs:

\[
\varphi \Rightarrow \alpha \quad \{\alpha\} \ s \ \{\beta\} \quad \beta \Rightarrow \psi \\
\{\varphi\} \ s \ \{\psi\}
\]

This is the only rule that is *not* directed entirely by syntax. This means a Hoare logic proof need not look like a derivation tree. Instead we can sprinkle assertions through our program and specially note uses of the consequence rule.
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\{ \text{True} \} \\
\begin{align*}
&i := 0; \\
&m := 1; \\
\text{while } i \neq N \text{ do} \\
&i := i + 1; \\
&m := m \times i \\
\text{od} \\
\{ m = N! \}
\]

\[
\{ \varphi \land e \} \ s_1 \ \{ \psi \} \quad \{ \varphi \land \neg e \} \ s_2 \ \{ \psi \} \\
\{ \varphi \} \ \text{if } e \ \text{then } s_1 \ \text{else } s_2 \ \text{fi} \ \{ \psi \} \\
\{ \varphi \} \ \text{while } e \ \text{do } s \ \text{od} \ \{ \varphi \land \neg e \} \\
\{ \varphi \} \ s_1 \ \{ \alpha \} \quad \{ \alpha \} \ s_2 \ \{ \psi \} \\
\{ \varphi \} \ s_1 ; s_2 \ \{ \psi \} \\
\varphi \Rightarrow \alpha \quad \{ \alpha \} \ s \ \{ \beta \} \quad \beta \Rightarrow \psi \\
\{ \varphi \} \ s \ \{ \psi \}
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{\text{True}\} & \quad \{i := 0;\} \quad \{m := 1;\} \\
\text{while } i \neq N \text{ do} & \quad \{i := i + 1;\} \\
\text{od} & \quad \{m := m \times i\} \\
\{m = i! \land i = N\} & \quad \{m = N!\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi \land e\} & \quad s_1 \quad \{\psi\} \quad \{\varphi \land \neg e\} \quad s_2 \quad \{\psi\} \\
\{\varphi\} & \quad \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi \quad } \{\psi\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi[x := e]\} & \quad x := e \quad \{\varphi\} \\
\{\varphi \land e\} & \quad s \quad \{\varphi\} \\
\{\varphi\} & \quad \text{while } e \text{ do } s \text{ od \quad } \{\varphi \land \neg e\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi\} & \quad s_1 \quad \{\alpha\} \quad \{\alpha\} \quad s_2 \quad \{\psi\} \\
\{\varphi\} & \quad s_1; s_2 \quad \{\psi\} \\
\varphi \Rightarrow \alpha & \quad \{\alpha\} \quad s \quad \{\beta\} \quad \beta \Rightarrow \psi \\
\{\varphi\} & \quad s \quad \{\psi\}
\end{align*}
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{\text{True}\} & : i := 0; \\
& m := 1; \\
\{m = i!\} & : \text{while } i \neq N \text{ do} \\
& i := i + 1; \\
& m := m \times i \\
\text{od} & : \{m = i! \land i = N\} \land \{m = N!\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi \land e\} & s_1 \{\psi\} \land \{\varphi \land \neg e\} s_2 \{\psi\} \\
\{\varphi\} & \text{ if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{\psi\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi[x := e]\} & x := e \{\varphi\} \\
\{\varphi \land e\} & s \{\varphi\} \\
\{\varphi\} & \text{ while } e \text{ do } s \text{ od } \{\varphi \land \neg e\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi\} & s_1 \{\alpha\} \land \{\alpha\} s_2 \{\psi\} \\
\{\varphi\} & s_1; s_2 \{\psi\}
\end{align*}
\]

\[
\begin{align*}
\varphi & \Rightarrow \alpha \\
\{\alpha\} & s \{\beta\} \land \beta \Rightarrow \psi
\end{align*}
\]

\[
\{\varphi\} s \{\psi\}
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{ \text{True} \} & \quad i := 0; \\
& \quad m := 1; \\
\{ m = i! \} & \quad \textbf{while } i \neq N \textbf{ do} \\
& \quad \quad i := i + 1; \\
& \quad \quad m := m \times i \\
\{ m = i! \} & \quad \textbf{od} \{ m = i! \land i = N \} \\
\{ m = N! \} &
\end{align*}
\]

\[
\begin{align*}
\{ \varphi \land e \} & \quad s_1 \quad \{ \psi \} \quad \{ \varphi \land \neg e \} & \quad s_2 \quad \{ \psi \} \\
\{ \varphi \} & \quad \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi} \quad \{ \psi \} \\
\{ \varphi \} & \quad \textbf{while } e \textbf{ do } s \textbf{ od} \quad \{ \varphi \land \neg e \} \\
\{ \varphi \} & \quad s_1 \quad \{ \alpha \} \quad \{ \alpha \} & \quad s_2 \quad \{ \psi \} \\
\{ \varphi \} & \quad s_1; s_2 \quad \{ \psi \} \\
\varphi \Rightarrow \alpha & \quad \{ \alpha \} \quad s \quad \{ \beta \} \quad \beta \Rightarrow \psi \\
\{ \varphi \} & \quad s \quad \{ \psi \}
\end{align*}
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{\text{True}\} & \quad \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{\psi\} \\
\{\varphi \land -e\} & \quad s_2 \{\psi\} \\
\{\varphi \land e\} & \quad s_1 \{\psi\} \\
\{\varphi\} & \quad \text{while } e \text{ do } s \od \{\varphi \land -e\} \\
\{\varphi\} & \quad s_1 \{\alpha\} \quad \{\alpha\} \quad s_2 \{\psi\} \\
\{\varphi\} & \quad s_1; s_2 \{\psi\} \\
\varphi & \Rightarrow \alpha \quad \{\alpha\} \quad s \{\beta\} \quad \beta \Rightarrow \psi \\
\{\varphi\} & \quad s \{\psi\}
\end{align*}
\]

\[\text{note: } (i+1)! = i! \times (i+1)\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{ \text{True} \} & \quad i := 0; \\
& \quad m := 1; \\
\{ m = i! \} & \quad \text{while } i \neq N \text{ do } \{ m = i! \land i \neq N \} \\
& \quad i := i + 1; \\
& \quad \{ m \times i = i! \} \\
& \quad m := m \times i \\
& \quad \{ m = i! \} \\
\text{od} & \quad \{ m = i! \land i = N \} \\
& \quad \{ m = N! \} \\
\end{align*}
\]

\[
\begin{align*}
\{ \varphi \land e \} s_1 \{ \psi \} & \quad \{ \varphi \land \neg e \} s_2 \{ \psi \} \\
\{ \varphi \} & \quad \text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } \{ \psi \} \\
\{ \varphi \} s \{ \varphi \} & \quad \{ \varphi \land e \} s \{ \varphi \} \\
\{ \varphi \} & \quad \text{while } e \text{ do } s \text{ od } \{ \varphi \land \neg e \} \\
\{ \varphi \} s_1 \{ \alpha \} & \quad \{ \alpha \} s_2 \{ \psi \} \\
& \quad \{ \varphi \} s_1; s_2 \{ \psi \} \\
\varphi \Rightarrow \alpha & \quad \{ \alpha \} s \{ \beta \} & \quad \beta \Rightarrow \psi \\
& \quad \{ \varphi \} s \{ \psi \}
\end{align*}
\]

\[
\text{note: } (i+1)! = i! \times (i+1)
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{&{\text{True}}\} \\
&\begin{cases} 
&i := 0; \\
&m := 1; \\
&\{m = i!\} \\
\textbf{while} \ i \neq N \ \textbf{do} \ {\{m = i! \land i \neq N\}} \\
&{\{m \times (i + 1) = (i + 1)!\}} \\
&i := i + 1; \\
&{\{m \times i = i!\}} \\
&m := m \times i \\
&\{m = i!\} \\
\textbf{od} \ {\{m = i! \land i = N\}} \\
&\{m = N!\} \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\{&\varphi \land e\} \ s_1 \ \{\psi\} \quad \{\varphi \land \neg e\} \ s_2 \ \{\psi\} \\
\{&\varphi\} \ \textbf{if} \ e \ \textbf{then} \ s_1 \ \textbf{else} \ s_2 \ \textbf{fi} \ \{\psi\} \\
\{&\varphi[x := e]\} \ x := e \ \{\varphi\} \\
\{&\varphi \land e\} \ s \ \{\varphi\} \\
\{&\varphi\} \ \textbf{while} \ e \ \textbf{do} \ s \ \textbf{od} \ \{\varphi \land \neg e\} \\
\{&\varphi\} \ s_1 \ \{\alpha\} \quad \{\alpha\} \ s_2 \ \{\psi\} \\
\{&\varphi\} \ s_1 ; s_2 \ \{\psi\} \\
\varphi \Rightarrow \alpha \quad \{\alpha\} \ s \ \{\beta\} \quad \beta \Rightarrow \psi \\
\{&\varphi\} \ s \ \{\psi\}
\end{align*}
\]

\[
\text{note: } (i + 1)! = i! \times (i + 1)
\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\{ \text{True} \}
\]
\[
i := 0;
\]
\[
m := 1;
\]
\[
\{ m = i! \}
\]
\[
\text{while } i \neq N \text{ do } \{ m = i! \land i \neq N \}
\]
\[
\{ m \times (i + 1) = (i + 1)! \}
\]
\[
i := i + 1;
\]
\[
\{ m \times i = i! \}
\]
\[
m := m \times i
\]
\[
\{ m = i! \}
\]
\[
\text{od } \{ m = i! \land i = N \}
\]
\[
\{ m = N! \}
\]

\[
\begin{align*}
\{ \varphi \land e \} & \rightarrow s_1 \{ \psi \} \quad \{ \varphi \land \neg e \} & \rightarrow s_2 \{ \psi \} \\
\text{fi } \{ \psi \}
\end{align*}
\]

\[
\begin{align*}
\{ \varphi[\times := e] \} & \rightarrow \times := e \{ \varphi \} \\
\text{while } e \text{ do } s \text{ od } \{ \varphi \land \neg e \}
\end{align*}
\]

\[
\begin{align*}
\{ \varphi \} & \rightarrow s_1 \{ \alpha \} \quad \{ \alpha \} & \rightarrow s_2 \{ \psi \} \\
\text{idem } s_1; s_2 \{ \psi \}
\end{align*}
\]

\[
\varphi \Rightarrow \alpha \quad \{ \alpha \} \rightarrow s \{ \beta \} \quad \beta \Rightarrow \psi
\]
\[
\{ \varphi \} \rightarrow s \{ \psi \}
\]

note: \((i + 1)! = i! \times (i + 1)\)
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{ \text{True} \} \\
& i := 0; \\
& m := 1; \{ m = i! \} \\
\{ m = i! \} \\
\textbf{while} \ i \neq N \ \textbf{do} \ \{ m = i! \land i \neq N \} \\
& \{ m \times (i + 1) = (i + 1)! \} \\
& i := i + 1; \\
& \{ m \times i = i! \} \\
& m := m \times i \\
& \{ m = i! \} \\
\textbf{od} \ \{ m = i! \land i = N \} \\
& \{ m = N! \} \\
\end{align*}
\]

note: \((i + 1)! = i! \times (i + 1)\)
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\{ \text{True} \}
\]
\[
i := 0;
\]
\[
\{ 1 = i! \} \ m := 1; \ { m = i! \}
\]
\[
\{ m = i! \}
\]
\[\textbf{while} \ i \neq N \ \textbf{do} \}
\[
\{ m = i! \land i \neq N \}
\]
\[
i := i + 1;
\]
\[
\{ m \times i = i! \}
\]
\[
\ m := m \times i
\]
\[
\{ m = i! \}
\]
\[\textbf{od} \}
\[
\ { m = i! \land i = N }\}
\]
\[
\{ m = N! \}
\]

\[
\{ \varphi \land e \} \ s_1 \ { \psi } \quad \{ \varphi \land \neg e \} \ s_2 \ { \psi }\]
\[
\{ \varphi \} \text{ if } e \text{ then } s_1 \text{ else } s_2 \text{ fi } { \psi }\]
\[
\{ \varphi [x := e] \} \ x := e \ { \varphi }\]
\[
\{ \varphi \land e \} \ s \ { \varphi }\]
\[
\{ \varphi \} \text{ while } e \text{ do } s \text{ od } { \varphi } \land \neg e\]
\[
\{ \varphi \} \ s_1 \ { \alpha } \quad \{ \alpha \} \ s_2 \ { \psi }\]
\[
\{ \varphi \} \ s_1 ; s_2 \ { \psi }\]
\[
\varphi \Rightarrow \alpha \quad \{ \alpha \} \ s \ { \beta } \quad \beta \Rightarrow \psi \quad \{ \varphi \} \ s \ { \psi }\]

\[
\text{note: } (i + 1)! = i! \times (i + 1)\]
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{ \text{True} \} \quad & i := 0; \{ 1 = i! \} \\
\{ 1 = i! \} \quad & m := 1; \{ m = i! \} \\
\{ m = i! \} \quad & \textbf{while} \ i \neq N \ \textbf{do} \quad \{ m = i! \land i \neq N \} \\
& \quad \{ m \times (i + 1) = (i + 1)! \} \\
& \quad i := i + 1; \quad \{ m \times i = i! \} \\
& \quad m := m \times i \quad \{ m = i! \} \\
\textbf{od} \quad & \{ m = i! \land i = N \} \\
\{ m = N! \} \\
\end{align*}
\]

\[\{ \varphi \land e \} \ s_1 \ \{ \psi \} \quad \{ \varphi \land \neg e \} \ s_2 \ \{ \psi \}\]

\[
\{ \varphi \} \ \text{if} \ e \ \text{then} \ s_1 \ \text{else} \ s_2 \ \text{fi} \ \{ \psi \}\]

\[
\{ \varphi \} \ \text{while} \ e \ \text{do} \ s \ \text{od} \ \{ \varphi \land \neg e \} \\
\{ \varphi \} \ s \ \{ \varphi \} \\
\{ \varphi \} \ s_1; s_2 \ \{ \psi \} \\
\varphi \Rightarrow \alpha \quad \{ \alpha \} \ s \ \{ \beta \} \quad \beta \Rightarrow \psi \quad \{ \varphi \} \ s \ \{ \psi \} \\
\]

note: \((i + 1)! = i! \times (i + 1)\)
Factorial Example

Let’s verify the Factorial program using our Hoare rules:

\[
\begin{align*}
\{\textbf{True}\} & \quad \{1 = 0!\} \ i := 0; \ \{1 = i!\} \\
\{\textbf{True}\} & \quad \{1 = i!\} \ m := 1; \ \{m = i!\} \\
\textbf{while} \ i \neq N \ \textbf{do} & \quad \{m = i! \land i \neq N\} \\
& \quad \{m \times (i + 1) = (i + 1)!\} \\
& \quad i := i + 1; \\
& \quad \{m \times i = i!\} \\
& \quad m := m \times i \\
\textbf{od} & \quad \{m = i! \land i = N\} \\
\{m = N!\} & \quad \{\textbf{True}\}
\end{align*}
\]

\[
\begin{align*}
\{\varphi \land e\} \ s_1 \ \{\psi\} & \quad \{\varphi \land \neg e\} \ s_2 \ \{\psi\} \\
\{\varphi\} & \quad \textbf{if} \ e \ \textbf{then} \ s_1 \ \textbf{else} \ s_2 \ \textbf{fi} \ \{\psi\} \\
\{\varphi[\times := e]\} \times := e \ \{\varphi\} \\
\{\varphi\} & \quad \textbf{while} \ e \ \textbf{do} \ s \ \textbf{od} \ \{\varphi \land \neg e\} \\
\{\varphi\} \ s_1 \ \{\alpha\} & \quad \{\alpha\} \ s_2 \ \{\psi\} \\
\{\varphi\} & \quad s_1; \ s_2 \ \{\psi\} \\
\varphi & \Rightarrow \alpha \\
\{\alpha\} \ s \ \{\beta\} & \quad \beta \Rightarrow \psi \\
\{\varphi\} & \quad s \ \{\psi\}
\end{align*}
\]

note: \((i + 1)! = i! \times (i + 1)\)