The small-step semantics of IMP
Arithmetic small-step semantics

\[ x/s \Rightarrow s(x) \quad \text{(Id)} \]

\[ \frac{n_3 = n_1 \circ n_2}{n_1 \circ n_2/s \Rightarrow n_3} \quad \text{(Diam)} \]

\[ \frac{a_1/s \Rightarrow a'_1}{a_1 \circ a_2/s \Rightarrow a_1 \circ a_2} \quad \text{(Diam-1)} \]

\[ \frac{\text{value}(v_1)}{v_1 \circ a_2/s \Rightarrow a_1 \circ a'_2} \quad \text{(Diam-2)} \]

where \[ \circ := + \mid - \mid \times \]
\[
\begin{align*}
\frac{a/s \Rightarrow a'}{(x ::= a)/s \Rightarrow (x ::= a')/s} & \quad (\text{Assign-Step}) \\
\frac{(x ::= n)/s \Rightarrow \text{SKIP}/s \& \{x \mapsto n\}}{} & \quad (\text{Assign}) \\
\frac{c_1/s \Rightarrow c'_1/s'}{(c_1;; c_2)/s \Rightarrow (c'_1;; c_2)/s'} & \quad (\text{Sequence-Step}) \\
\frac{(\text{SKIP};; c_2)/s \Rightarrow c_2/s}{(\text{Sequence-Finish})} & \\
\frac{\text{IFB true THEN } c_1 \text{ ELSE } c_2 \text{ FI}/s \Rightarrow c_1/s}{(\text{If-True})} & \quad \frac{\text{IFB false THEN } c_1 \text{ ELSE } c_2 \text{ FI}/s \Rightarrow c_2/s}{(\text{If-False})} \\
\frac{b/s \Rightarrow b'}{\text{IFB } b \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI}/s \Rightarrow \text{IFB } b' \text{ THEN } c_1 \text{ ELSE } c_2 \text{ FI}/s} & \quad (\text{If-Step}) \\
\frac{\text{WHILE } b \text{ DO } c \text{ END}}{c' = c;; \text{WHILE } b \text{ DO } c \text{ END}/s \Rightarrow \text{IFB } b \text{ THEN } c' \text{ ELSE } \text{SKIP} \text{ FI}/s} & \quad (\text{While})
\end{align*}
\]
Factorial: 3!

State: {}

\[ X := 3; ; \]
\[ Z := X; ; \]
\[ Y := 1; ; \]
\[ \text{WHILE } ! (Z = 0) \text{ DO} \]
\[ \quad Y := Y \times Z; ; \]
\[ \quad Z := Z - 1 \]
\[ \text{END} \]

Which rules?
Factorial: 3!

State: {} 

X ::= 3;;
Z ::= X;;
Y ::= 1;;
WHILE ! (Z = 0) DO
    Y ::= Y * Z;;
    Z ::= Z - 1
END

Which rules?
Sequence-Step, Assign
Factorial: 3!

State: \( \{ X = 3 \} \)

```
SKIP;;
Z ::= X;;
Y ::= 1;;
WHILE ! (Z = 0) DO
    Y ::= Y \* Z;;
    Z ::= Z - 1
END
```

Which rules?
Factorial: 3!

State: \{X = 3\}

```
SKIP;;
Z ::= X;;
Y ::= 1;;
WHILE ! (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
END
```

- Which rules?

Sequence-Finish
Factorial: 3!

State: \( \{ X = 3 \} \)

\[
\begin{align*}
Z & := X;; \\
Y & := 1;; \\
\text{WHILE} \quad ! (Z = 0) \quad \text{DO} \\
& \quad Y := Y \times Z;; \\
& \quad Z := Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?
Factorial: 3!

State: \( \{X = 3\} \)

\[
\begin{align*}
Z & := X;; \\
Y & := 1;; \\
\text{WHILE} \quad & (Z \neq 0) \quad \text{DO} \\
& Y := Y \times Z; \\
& Z := Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?

- Sequence-Step
- Assign-Step
- Id
Factorial: 3!

State: \( \{X = 3\} \)

\[
\begin{align*}
Z & := 3 ;; \\
Y & := 1 ;; \\
\text{WHILE} & \ ! (Z = 0) \ \text{DO} \\
& \quad Y := Y * Z ;; \\
& \quad Z := Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?
Factorial: 3!

State: \{X = 3\}

\[
\begin{align*}
Z & := 3 ;; \\
Y & := 1 ;; \\
\text{WHILE} & ! (Z = 0) \text{ DO} \\
& \quad Y := Y \times Z ;; \\
& \quad Z := Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?

Sequence-Step, Assign
Factorial: 3!

State: \{X = 3, Z = 3\}

```plaintext
SKIP;;
Y ::= 1;;
WHILE ! (Z = 0) DO
    Y ::= Y * Z;;
    Z ::= Z - 1
END
```

Which rules?
Factorial: 3!

State: \{X = 3, Z = 3\}

```plaintext
SKIP;;
Y ::= 1;;
WHILE !(Z = 0) DO
    Y ::= Y * Z;;
    Z ::= Z - 1
END
```

Which rules?

Sequence-Finish
Factorial: 3!

State: \( \{ X = 3, Z = 3 \} \)

\[
\begin{align*}
Y & := 1 ;; \\
\text{WHILE} \! (Z = 0) & \text{ DO} \\
& \quad Y := Y \times Z ;; \\
& \quad Z := Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?
Factorial: 3!

State: \(\{X = 3, \ Z = 3\}\)

\[
\begin{align*}
Y & ::= 1 ; ; \\
\text{WHILE} \ ! (Z = 0) \ \text{DO} \\
& \quad Y ::= Y \times Z ; ; \\
& \quad Z ::= Z - 1 \\
\text{END}
\end{align*}
\]

Which rules?

Sequence-Step, Assign
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

```plaintext
SKIP;;
WHILE ! (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
END
```

Which rules?
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

```plaintext
SKIP;;
WHILE ! (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
END
```

Which rules?

Sequence-Finish
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

```c
WHILE !(Z = 0) DO
    Y ::= Y * Z;;
    Z ::= Z - 1
END
```

Which rules?
Factorial: 3!

State: \{X = 3, Z = 3, Y = 1\}

```
WHILE ! (Z = 0) DO
    Y ::= Y * Z;;
    Z ::= Z - 1
END
```

<table>
<thead>
<tr>
<th>Which rules?</th>
</tr>
</thead>
<tbody>
<tr>
<td>While</td>
</tr>
</tbody>
</table>
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

```
IFB ! (Z = 0) THEN
    Y ::= Y * Z;;
    Z ::= Z - 1;;
    WHILE ! (Z = 0) DO
        Y ::= Y * Z;;
        Z ::= Z - 1
    END
ELSE
    SKIP
FI
```

Which rules?
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

\[
\begin{align*}
\text{IFB} & \quad \neg (Z = 0) \quad \text{THEN} \\
& \quad Y ::= Y \times Z ;; \\
& \quad Z ::= Z - 1 ;; \\
& \quad \text{WHILE} \quad \neg (Z = 0) \quad \text{DO} \\
& \quad \quad Y ::= Y \times Z ;; \\
& \quad \quad Z ::= Z - 1 \\
& \quad \text{END} \\
\text{ELSE} \\
& \quad \text{SKIP} \\
\text{FI}
\end{align*}
\]

Which rules?

If-Step, Not-Step, Eq-1, Id
Factorial: 3!

State: \(\{X = 3, Z = 3, Y = 1\}\)

\[
\text{IFB } \neg (3 = 0) \text{ THEN} \\
\begin{align*}
Y & := Y \times Z; \\
Z & := Z - 1;
\end{align*}
\text{WHILE } \neg (Z = 0) \text{ DO} \\
\begin{align*}
Y & := Y \times Z; \\
Z & := Z - 1
\end{align*}
\text{END} \\
\text{ELSE} \\
\text{SKIP}
\text{FI}
\]

Which rules?
Factorial: 3!

State: \{X = 3, Z = 3, Y = 1\}

\[
\text{IFB } ! \ (3 = 0) \ \text{THEN} \\
\quad \text{Y ::= } Y \times Z;; \text{\hspace{1cm} \text{Y ::= } Y \times Z;;} \\
\quad \text{Z ::= } Z - 1;; \text{\hspace{1cm} \text{Z ::= } Z - 1} \\
\quad \text{WHILE } ! \ (Z = 0) \ \text{DO} \\
\qquad \text{Y ::= } Y \times Z;; \\
\qquad \text{Z ::= } Z - 1 \\
\quad \text{END} \\
\text{ELSE} \\
\quad \text{SKIP} \\
\text{FI}
\]

- Which rules?

If-Step, Not-Step, Eq
Factorial: 3!

State: \{X = 3, Z = 3, Y = 1\}

IFB ! false THEN
Y ::= Y * Z;;
Z ::= Z - 1;;
WHILE ! (Z = 0) DO
Y ::= Y * Z;;
Z ::= Z - 1
END
ELSE
SKIP
FI

Which rules?
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

```plaintext
IFB ! false THEN
    Y ::= Y * Z;;
    Z ::= Z - 1;;
    WHILE ! (Z = 0) DO
        Y ::= Y * Z;;
        Z ::= Z - 1
    END
ELSE
    SKIP
FI
```

Which rules?

If-Step, Not-False
Factorial: 3!

State: \( \{X = 3, Z = 3, Y = 1\} \)

\[
\begin{align*}
\text{IFB true THEN} & \\
Y & := Y \ast Z ;; \\
Z & := Z - 1 ;; \\
\text{WHILE } ! (Z = 0) \text{ DO} & \\
Y & := Y \ast Z ;; \\
Z & := Z - 1 \\
\text{END} & \\
\text{ELSE} & \\
\text{SKIP} & \\
\text{FI} & \\
\end{align*}
\]

Which rules?
Factorial: 3!

State: \{X = 3, Z = 3, Y = 1\}

\[
\text{IFB true THEN}
\begin{align*}
Y & := Y \times Z ;; \\
Z & := Z - 1 ;; \\
\text{WHILE } ! (Z = 0) \text{ DO} \\
Y & := Y \times Z ;; \\
Z & := Z - 1
\end{align*}
\text{END}
\]

ELSE
SKIP
FI

Which rules?

If-True
Factorial: 3!

State: \{X = 3, \ Z = 3, \ Y = 1\}

Y ::= Y \times Z;;
Z ::= Z - 1;;
WHILE ! (Z = 0) DO
  Y ::= Y \times Z;;
  Z ::= Z - 1
END

To be continued...
Non-terminating programs

```
WHILE true DO
    SKIP
END
```

Which rules?
Non-terminating programs

```
WHILE true DO
  SKIP
END
```

Which rules?

While
Non-terminating programs

```plaintext
IFB true DO
  SKIP;;
  WHILE true DO
    SKIP
    END
ELSE
  SKIP
FI
```

Which rules?
Non-terminating programs

```plaintext
IFB true DO
    SKIP;;
WHILE true DO
    SKIP
    END
ELSE
    SKIP
FI
```

Which rules?

If-True
Non-terminating programs

```plaintext
SKIP;;
WHILE true DO
  SKIP
END
```

Which rules?
Non-terminating programs

```
SKIP;;
WHILE true DO
  SKIP
END
```

- Which rules?
  - Sequence-Finish
Non-terminating programs

```pseudocode
WHILE true DO
  SKIP
END
```

We are back to where we started!
Small-step semantics of IMP

- We have seen how reduction of IMP programs unfolds
- We have seen how small-step semantics behaves with non-terminating computations
Adding Concurrency
Edsger W. Dijkstra introduced the construct `parbegin/parend` in 1965, ([EDW123, Section 2.1](#)) as an extension of ALGOL 60, to introduce mutual exclusion, the need for synchronization primitives (semaphores), and bounded buffers.

This work was published before the birth of parallel computing and before computer networks were in operation.

- **ILLIAC IV**, the first parallel super computer, became operational in 1972.
- Dartmouth Time-Sharing System (DTSS) starts operation in 1964, the first successful large-scale time-sharing system.
- The first four nodes of ARPANET appear in 1969.
Adding concurrency

\[ c ::= \cdots | \text{PAR } c_1 \text{ WITH } c_2 \text{ END } | \cdots \]

\[
\frac{c_1/s \Rightarrow c_1'/s'}{
\text{PAR } c_1 \text{ WITH } c_2 \text{ END }/s \Rightarrow \text{PAR } c_1'/\text{WITH } c_2 \text{ END }/s'} \tag{Par-1}
\]

\[
\frac{c_2/s \Rightarrow c_2'/s}{
\text{PAR } c_1 \text{ WITH } c_2 \text{ END }/s \Rightarrow \text{PAR } c_1 \text{ WITH } c_2'/\text{END }/s'} \tag{Par-2}
\]

\[
\frac{c_2'/s \Rightarrow c_2'/s}{
\text{PAR } c_1 \text{ WITH } c_2 \text{ END }/s \Rightarrow \text{PAR } c_1 \text{ WITH } c_2'/\text{END }/s'} \tag{Par-Done}
\]

\[
\text{PAR SKIP WITH SKIP END }/s \Rightarrow \text{SKIP }/s
\]
Example

Let \( X = 0 \). What is the value of \( X \) after running this program?

\[
\begin{align*}
\text{PAR} \\
\quad X &::= X + 1 \\
\text{WITH} \\
\quad X &::= X + 1 \\
\text{END}
\end{align*}
\]

Which rules?
Example

Let $X=0$. What is the value of $X$ after running this program?

```
PAR
  X ::= X + 1
WITH
  X ::= X + 1
END
```

Which rules?

- Par-1, Assign-Step, Id
- Par-2, Assign-Step, Id
Example

Store: \{x = 0\}

PAR
  \(x ::= 0 + 1\)
WITH
  \(x ::= x + 1\)
END

Which rules?

- Par-1, Assign-Step, Plus
- Par-2, Assign-Step, Id
Example

Store: \( \{ x = 0 \} \)

```
PAR
  x ::= 1
WITH
  x ::= x + 1
END
```

Which rules?

- Par-1, Assign
- Par-2, Assign-Step, Id

What are the valid outcomes?
Example

Store: \( \{X = 0\} \)

\[
\begin{align*}
\text{PAR} \\
X &::= 1 \\
\text{WITH} \\
X &::= X + 1 \\
\text{END}
\end{align*}
\]

Which rules?

- Par-1, Assign
- Par-2, Assign-Step, Id

What are the valid outcomes?

\( X = 1 \) or \( X = 2 \)
Data-races

When one write to $X$ can run concurrently with a read/write to $X$, we say that there is a \textit{data-race}.

Data-races are a source of \textit{unexpected} non-determinism and are therefore considered to be an \textit{error}.
Example

In the following example, the write to \( X \) always executes before the reads from \( X \) (they are ordered), which means that there are no data-races in the program. Is it deterministic?

\[
\begin{align*}
X & := 1 ;; \\
\text{PAR} & \\
& \quad Y := X + 1 \\
\text{WITH} & \\
& \quad Z := X + 1 \\
\text{END} & 
\end{align*}
\]
Example

In the following example, the write to $X$ always executes before the reads from $X$ (they are ordered), which means that there are no data-races in the program. Is it deterministic?

```
X ::= 1 ;;
PAR
  Y ::= X + 1
WITH
  Z ::= X + 1
END
```

In this language the *only* source of non-determinism is a data-race!
If data-races should be avoided, then how to parallel tasks communicate?
Adding synchronization

\[ c ::= \cdots \mid \text{NEXT} \mid \cdots \]

\[
\text{PAR} \ 	ext{NEXT}; \ ;t_1\text{WITH}\ NEXT; \ ;t_2\ \text{END}/s \Rightarrow \text{PAR}\ t_1\ \text{WITH}\ t_2\ \text{END}/s \]

(Sync)
Example

Let $X=0$. What is the value of $X$ after running this program?

```par
PAR
  X ::= X + 1;;
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?
Example

Let $X=0$. What is the value of $X$ after running this program?

```
PAR
  X ::= X + 1;;
NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?

Par-1, Assign-Step, Id
Example

Memory: \{X = 0\}

```plaintext
PAR
  X ::= 0 + 1;;
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?
Example

Memory: \(\{X = 0\}\)

\[
\begin{align*}
\text{PAR} & \quad X ::= \mathit{0} + 1;; \\
\text{NEXT} & \\
\text{WITH} & \quad \text{NEXT};; \\
X ::= X + 1 & \\
\text{END} &
\end{align*}
\]

Which rules?

Par-1, Assign-Step, Plus
Example

Memory: \{x = 0\}

\begin{verbatim}
PAR
  \texttt{X ::= 1;;}
  \texttt{NEXT}
WITH
  \texttt{NEXT;;}
  \texttt{X ::= X + 1}
END
\end{verbatim}

Which rules?
Example

Memory: \( \{X = 0\} \)

```
PAR
  X ::= 1;;
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?
Par-1, Assign
Example

Memory: \( \{X = 1\} \)

```
PAR
  SKIP;;
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?
Example

Memory: \{ X = 1 \}

```
PAR
  SKIP;;
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
```

Which rules?

Par-1, Sequence-Finish
Example

Memory: \{X = 1\}

PAR
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END

Which rules?
Example

Memory: \{X = 1\}

\begin{verbatim}
PAR
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END
\end{verbatim}

Which rules?

We are stuck!
Adding synchronization

\[ c ::= \cdots \mid \text{NEXT} \mid \cdots \]

\[ \text{PAR NEXT;} ; t_1 \text{WITH NEXT;} ; t_2 \text{ END} / s \Rightarrow \text{PAR } t_1 \text{ WITH } t_2 \text{ END} / s \]  
\text{(Sync)}

\[ t_1 \equiv t'_1 \quad t'_1 / s \Rightarrow t'_2 / s' \quad t_2 \equiv t'_2 \]
\[ t_1 / s \Rightarrow t_2 / s' \]  
\text{(Congr)}

\[ t \equiv t ; \text{SKIP} \quad t \equiv t \]

\[ \text{PAR } t_1 \text{WITH } t_2 \text{ END } \equiv \text{PAR } t'_1 \text{WITH } t'_2 \text{ END} \]
\text{(C-skip,C-refl,C-par)}
Example

Memory: \{X = 1\}

PAR
  NEXT
WITH
  NEXT;;
  X ::= X + 1
END

Which rules?
Example

Memory: \{X = 1\}

\begin{verbatim}
PAR
  NEXT
WITH
    NEXT;;
    X ::= X + 1
END
\end{verbatim}

Which rules?

Congr, C-par, C-skip, C-refl, Sync
Example

Memory: \( \{X = 1\} \)

```
PAR
  SKIP
WITH
  X ::= X + 1
END
```

What is the value of \(X\) after running this program?
Example
Memory: \{X = 1\}

```
PAR
  SKIP
WITH
    X ::= X + 1
END
```

What is the value of \(X\) after running this program? \(X = 2\). The program is deterministic.
Example

PAR
  NEXT;;
  SKIP
WITH
  SKIP
END

Which rules?
Which rules? **We are stuck.** One task is *deadlocked!*

This language does **not** enjoy progress because of NEXT. The semantics are no longer normalizing.
Theorem step_deterministic:
deterministic step.

Theorem strong_progress : \( \forall t, \text{value } t \lor (\exists t', t \Rightarrow t') \).

Lemma value_is_nf : \( \forall v, \text{value } v \Rightarrow \text{normal_form step } v \).

Lemma nf_is_value : \( \forall t, \text{normal_form step } t \Rightarrow \text{value } t \).

Theorem step_normalizing :
normalizing step.
Theorem step_deterministic:
  deterministic step.

Theorem strong_progress : forall t,
  value t ∨ (exists t', t ⇒ t').

Lemma value_is_nf : forall v,
  value v → normal_form step v.

Lemma nf_is_value : forall t,
  normal_form step t → value t.

Theorem step_normalizing :
  normalizing step.