CS720

Logical Foundations of Computer Science

Lecture 18: Imperative and parallel semantics

Tiago Cogumbreiro

The small-step semantics of IMP

Arithmetic small-step semantics

$$\frac{}{x/s \Rightarrow s(x)}\text{(Id)}$$

$$\frac{n_3 = n_1 \ \diamond \ n_2}{n_1 \ \diamond \ n_2/s \Rightarrow n_3}\text{(Diam)} \qquad \frac{a_1/s \Rightarrow a_1'}{a_1 \ \diamond \ a_2/s \Rightarrow a_1 \diamond a_2}\text{(Diam-1)}$$

$$\frac{\text{value}(v_1) \qquad a_2 \Rightarrow a_2'}{v_1 \ \diamond \ a_2/s \Rightarrow a_1 \diamond a_2'}\text{(Diam-2)}$$

where $\diamond ::= + \mid - \mid \times$

$$\frac{a/s \Rightarrow a'}{(x ::= a)/s \Rightarrow (x ::= a')/s}\text{(Assign-Step)} \qquad \frac{}{(x ::= n)/s \Rightarrow \mathtt{SKIP}/s\&\{x \mapsto n\}}\text{(Assign)}$$

$$\frac{c_1/s \Rightarrow c_1'/s'}{(c_1;;c_2)/s \Rightarrow (c_1';;c_2)/s'}\text{(Sequence-Step)} \qquad \frac{}{(\mathtt{SKIP};;c_2)/s \Rightarrow c_2/s}\text{(Sequence-Finish)}$$

$$\frac{}{\mathtt{IFB}\ \mathtt{true}\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2\ \mathtt{FI}/s \Rightarrow c_1/s}\text{(If-True)} \qquad \frac{}{\mathtt{IFB}\ \mathtt{false}\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2\ \mathtt{FI}/s \Rightarrow c_2/s}\text{(If-False)}$$

$$\frac{b/s \Rightarrow b'}{\mathtt{IFB}\ b\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2\ \mathtt{FI}/s \Rightarrow \mathtt{IFB}\ b'\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2\ \mathtt{FI}/s}\text{(If-Step)}$$

$$\frac{c' = c;;\mathtt{WHILE}\ b\ \mathtt{DO}\ c\ \mathtt{END}}{\mathtt{WHILE}\ b\ \mathtt{DO}\ c\ \mathtt{END}/s \Rightarrow \mathtt{IFB}\ b\ \mathtt{THEN}\ c'\ \mathtt{ELSE}\ \mathtt{SKIP}\ \mathtt{FI}/s}\text{(While)}$$

Factorial: 3!

State: $\{\}$

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

```
Z ::= X;;
Y ::= 1;;
WHILE ! (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
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-Step, Id

Factorial: 3!

State: {X = 3}

```
Z ::= 3;;
Y ::= 1;;
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
```

> Which rules?

Sequence-Step, Assign

Factorial: 3!

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

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

> Which rules?

Factorial: 3!

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

```
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}`

```
Y ::= 1;;
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
```

> Which rules?

Sequence-Step, Assign

Factorial: 3!

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

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

> Which rules?

Factorial: 3!

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

```
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}

```
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
```

> Which rules?

While

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`}

```
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?

If-Step, Not-Step, Eq-1, Id

Factorial: 3!

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

```
IFB ! (3 = 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`}

```
IFB ! (3 = 0) 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-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}

```
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`}

```
IFB true 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}

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

> Which rules?

If-True

Factorial: 3!

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

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

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

> Which rules?

Non-terminating programs

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

> Which rules?

If-True

Non-terminating programs

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

> Which rules?

Non-terminating programs

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

> Which rules?

Sequence-Finish

Non-terminating programs

```
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

Let us add a `par` construct

> 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.
- The field of concurrency theory is born in the 1960's Notably, Carl Adam Petri's PhD thesis is published in 1963.

Adding concurrency

$$c ::= \cdots \mid \texttt{PAR}\ c_1\ \texttt{WITH}\ c_2\ \texttt{END} \mid \cdots$$

$$\frac{c_1 / s \Rightarrow c_1' / s'}{\texttt{PAR}\ c_1\ \texttt{WITH}\ c_2\ \texttt{END} / s \Rightarrow \texttt{PAR}\ c_1'\ \texttt{WITH}\ c_2\ \texttt{END} / s'}\ \text{(Par-1)}$$

$$\frac{c_2 / s \Rightarrow c_2' / s}{\texttt{PAR}\ c_1\ \texttt{WITH}\ c_2\ \texttt{END} / s \Rightarrow \texttt{PAR}\ c_1\ \texttt{WITH}\ c_2'\ \texttt{END} / s'}\ \text{(Par-2)}$$

$$\frac{}{\texttt{PAR SKIP WITH SKIP END} / s \Rightarrow \texttt{SKIP} / s}\ \text{(Par-Done)}$$

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?

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}`

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

> 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 *data-race*.

Data-races are a source of unexpected non-determinism and are therefore considered to be an **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?**

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

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

$$c ::= \cdots \mid \texttt{NEXT} \mid \cdots$$

$$\frac{}{\texttt{PAR}\ \texttt{NEXT};\ ; t_1 \texttt{WITH}\ \texttt{NEXT};\ ; t_2\ \texttt{END}/s \Rightarrow \texttt{PAR}\ t_1\ \texttt{WITH}\ t_2\ \texttt{END}/s}\,(\text{Sync})$$

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?

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}`

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

> Which rules?

Example

Memory: `{X = 0}`

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

> Which rules?

Par-1, Assign-Step, Plus

Example

Memory: {X = 0}

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

> 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}`

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

> Which rules?

We are stuck!

Adding synchronization

$$c ::= \cdots \mid \texttt{NEXT} \mid \cdots$$

$$\frac{}{\texttt{PAR}\ \texttt{NEXT};;t_1 \texttt{WITH}\ \texttt{NEXT};;t_2\ \texttt{END}/s \Rightarrow \texttt{PAR}\ t_1\ \texttt{WITH}\ t_2\ \texttt{END}/s}\text{(Sync)}$$

$$\frac{t_1 \equiv t_1' \qquad t_1'/s \Rightarrow t_2'/s' \qquad t_2 \equiv t_2'}{t_1/s \Rightarrow t_2/s'}\text{(Congr)}$$

$$t \equiv t;;\texttt{SKIP} \qquad t \equiv t \qquad \frac{t_1 \equiv t_1' \qquad t_2 \equiv t_2'}{\texttt{PAR}\ t_1 \texttt{WITH}\ t_2\ \texttt{END} \equiv \texttt{PAR}\ t_1' \texttt{WITH}\ t_2'\ \texttt{END}}$$

(C-skip,C-refl,C-par)

Example

Memory: {X = 1}

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

> Which rules?

Example

Memory: `{X = 1}`

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

> 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?

Example

```
PAR
  NEXT;;
  SKIP
WITH
  SKIP
END
```

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

This language does **not** enjoy progress because of `NEXT`. The semantics are no longer normalizing.

Workshop

```
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.
```

Workshop

```
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.
```