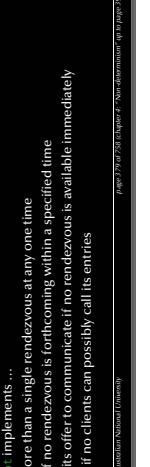
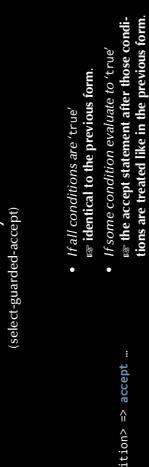




<h2>Non-determinism</h2> <h3>Selective Synchronization</h3> <h4>Message-based selective synchronization in Ada</h4> <p>Forms of selective waiting:</p> <pre>select_statement ::= select   [guard] selective_accept alternative   { or [guard] selective_accept alternative   { else sequence_of_statements }   end select;</pre> <p>guard ::= when &lt;condition&gt; =&gt; selective_accept_alternative ::= accept_alternative   delay_alternative   terminate_alternative</p> <pre>accept_alternative ::= accept_statement [ sequence_of_statements ] delay_alternative ::= delay_statement [ sequence_of_statements ] terminate_alternative ::= terminate;</pre> <p>accept_statement ::= accept entry direct_name [(entry_index)] parameter_profile [do handled_sequence_of_statements end entry Identifier];</p> <pre>delay_statement ::= delay until delay_expression;   delay delay_expression;</pre>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>	<h2>Non-determinism</h2> <h3>Selective Synchronization</h3> <h4>Message-based selective synchronization in Ada</h4> <p>Forms of selective waiting:</p> <pre>select_statement ::= select   [guard] selective_accept alternative   { or [guard] selective_accept alternative   { else sequence_of_statements }   end select;</pre> <p>guard ::= when &lt;condition&gt; =&gt; selective_accept_alternative ::= accept_alternative   delay_alternative   terminate_alternative</p> <pre>accept_alternative ::= accept_statement [ sequence_of_statements ] delay_alternative ::= delay_statement [ sequence_of_statements ] terminate_alternative ::= terminate;</pre> <p>accept_statement ::= accept entry direct_name [(entry_index)] parameter_profile [do handled_sequence_of_statements end entry Identifier];</p> <pre>delay_statement ::= delay until delay_expression;   delay delay_expression;</pre>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>
<h2>Non-determinism</h2> <h3>Selective Synchronization</h3> <h4>Basic forms of selective synchronization</h4> <p>... underlying concept: Dijkstra's guarded commands</p> <pre>selective_accept implements ...   ... wait for more than a single rendezvous at any one time   ... time-out if no rendezvous is forthcoming within a specified time   ... withdraw its offer to communicate it/no rendezvous is available immediately   ... terminate if no clients can possibly call its entries</pre>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>	<h2>Non-determinism</h2> <h3>Selective Synchronization</h3> <h4>Basic forms of selective synchronization</h4> <p>... underlying concept: Dijkstra's guarded commands</p> <pre>selective_accept implements ...   ... wait for more than a single rendezvous at any one time   ... time-out if no rendezvous is forthcoming within a specified time   ... withdraw its offer to communicate it/no rendezvous is available immediately   ... terminate if no clients can possibly call its entries</pre>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>
<p>383</p>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>	<p>386</p>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>
<p>387</p>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>	<p>388</p>	 <p>© 2020, Dr. R. Zimmer, The Australian National University</p>

**Non-determinism**

**Selective Synchronization**

**Basic forms of selective synchronization**

(select-accept)

- If none of the entries have waiting calls  
  ⇒ the process is suspended until a call arrives.
- If exactly one of the entries has waiting calls  
  ⇒ this entry is selected.
- If multiple entries have waiting calls
  - one of them is selected non-deterministically. The selection is based on the real-time systems annex.
  - The code following the selected entry (if any) is executed and the select statement completes.

© 2020 IFRP, F. Zimmer, The Australian National University

Page 387 of 50 | Chapter 4 - Non-determinism (pp 349-375)

---

**Non-determinism**

**Selective Synchronization**

**Basic forms of selective synchronization**

(select-guarded-accept-delay)

- If none of the open entities have waiting calls before the deadline specified by the earliest open alternative  
  ⇒ this earliest delay alternative is chosen and the entities associated with it are executed.
- Otherwise, one of the open entities with waiting calls is chosen as above.
- This enables a task to withdraw its offer to accept a set of calls if no other task is calling after some time.

© 2020 IFRP, F. Zimmer, The Australian National University

Page 388 of 50 | Chapter 4 - Non-determinism (pp 349-375)

---

**Non-determinism**

**Selective Synchronization**

**Conditional entry-calls**

```
conditional_entry_call ::= 
  select
    entry_call statements
  [sequence_of_statements]
  else
    sequence_of_statements
  end select;
```

Example:

```
select
  Light_Monitor_WAIT_for_Light;
  Lux := True;
else;
```

© 2020 IFRP, F. Zimmer, The Australian National University

Page 389 of 50 | Chapter 4 - Non-determinism (pp 349-375)

**Non-determinism**

### Selective Synchronization

#### Timed entry-calls

```
timed_entry_call ::= 
  select
    entry_call_statement
  [Sequence_of_statements]
  or
  delay_alternative
end_select;

Example:
select
  Controller.Request (SomeItem);
  ----- process data
or
  delay 45.0; ----- seconds
  ----- try something else
end_select;
```

© 2020 University of Twente, The Australian National University

Page 389 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Non-deterministic programs

**Partial correctness:**  $(P(I) \wedge \text{terminates}(\text{Program}(I, O))) \Rightarrow Q(I, O)$

**Total correctness:**  $P(I) \Rightarrow (\text{terminates}(\text{Program}(I, O)) \wedge Q(I, O))$

**Safety properties:**  $(P(I) \wedge \text{Processes}(I, S)) \Rightarrow \Box Q(I, S)$   
where  $\Box Q$  means that  $Q$  does always hold

**Liveness properties:**  $(P(I) \wedge \text{Processes}(I, S)) \Rightarrow \Diamond Q(I, S)$   
where  $\Diamond Q$  means that  $Q$  does eventually hold (and will then stay true)

and  $S$  is the current state of the concurrent system

© 2020 University of Twente, The Australian National University

Page 390 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Non-deterministic programs

**Concrete:**  
Every time you formulate a non-deterministic statement like the one on the left you need to formulate an invariant which holds true whenever alternative will actually be chosen.  
This is very similar to finding top invariants in sequential programs

```
select
  when <condition> => accept ...
  or
  when <condition> => accept ...
  or
  when <condition> => accept ...
  -
  end select;
```

© 2020 University of Twente, The Australian National University

Page 391 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Message-based selective synchronization in Ada

#### Selective Synchronization

Forms of selective waiting:

```
select_statement ::= selective_accept
  conditional_entry_call_1
  |
  timed_entry_call;
asynchronous_select;
```

... underlying concept: Dijkstra's guarded commands

asynchronous\_select implements ...  
the possibility to escape a running code block due to an event from outside this task.  
(outside the scope of this course) check Real-Time Systems

© 2020 University of Twente, The Australian National University

Page 390 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Correctness of non-deterministic programs

For example (in verbal form):  
“Mutual exclusion accessing a specific resource holds true for all possible numbers of sequences or interleavings of requests to it”

An invariant would for instance be that the number of writing tasks inside a protected object is less or equal to one.

Those invariants are the only practical way to guarantee (in a logical sense) correctness in concurrent/ non-deterministic systems.  
(as enumerating all possible cases and proving them individually is in general not feasible)

Therefore correctness predicates need to be based on invariants,  
i.e. invariant predicates which are independent of the potential execution sequences,  
yet support the overall correctness predicates.

© 2020 University of Twente, The Australian National University

Page 391 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Summary

**Non-Determinism**

- Non-determinism by design:**
  - Benefits & considerations
- Non-determinism by interaction:**
  - Selective synchronization
  - Selective accepts
  - Selective calls
- Correctness of non-deterministic programs:**
  - Sources of non-determinism
  - Predicates & invariants

© 2020 University of Twente, The Australian National University

Page 392 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Sources of Non-determinism

As concurrent entities are not in “lock-step” synchronization, they “overtake” each other and arrive at synchronization points in non-deterministic order, due to (just a few):

- Operating systems / runtime environments:  
↳ Schedulers are often non-deterministic.  
↳ Message passing systems: react load depended.
- Networks & communication systems:  
↳ Traffic will arrive in an unpredictable way (non-deterministic).  
↳ Communication systems congestions are generally unpredictable.
- Computing hardware:  
↳ Timers drift and clocks have granularities.  
↳ Processors have out-of-order units.
- ... basically: Physical systems (and computer systems connected to the physical world)  
are inherently non-deterministic.

© 2020 University of Twente, The Australian National University

Page 393 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

**Non-determinism**

### Correctness of non-deterministic programs

For example (in verbal form):  
“Mutual exclusion accessing a specific resource holds true for all possible numbers of sequences or interleavings of requests to it”

An invariant would for instance be that the number of writing tasks inside a protected object is less or equal to one.

Those invariants are the only practical way to guarantee (in a logical sense) correctness in concurrent/ non-deterministic systems.  
(as enumerating all possible cases and proving them individually is in general not feasible)

Therefore correctness predicates need to be based on invariants,  
i.e. invariant predicates which are independent of the potential execution sequences,  
yet support the overall correctness predicates.

© 2020 University of Twente, The Australian National University

Page 394 of 393 Chapter 4: Non-determinism (IP: 192.168.1.1)

