Non-determinism

**Non-determinism by design**

- Selection is non-deterministic for any \( x \leq y \rightarrow m := x \) if the selection condition \( x \leq y \) is evaluated to true and the alternative is not predefined. The compiler/ runtime environment can directly translate the source code into a concurrent implementation.

- All true case statements in any language are potentially concurrent and non-deterministic.

**Non-determinism by interaction**

- The boolean expressions are local expressions, i.e., if none of them evaluates to true, the tuple is suspended until the next activity on one of the channels.

- Any Occam process can be employed in the ALT statement.

**Non-determinism in Occam2**

1. **Guard**
   - Process Selection in Occam2

2. **Guard**
   - Process Selection in Occam2

**Non-determinism in Occam2**

- Guards are referring to boolean expressions and/or channel input operations.
- The boolean expressions are local expressions, i.e., if none of them evaluate to true, then the process is stopped.
- If all triggered channel input operations evaluate to false, the process is suspended until subsequent activity on one of the channels.
- Any Occam process can be employed in the ALT statement.
- The ALT statement is non-deterministic (there is also a deterministic version PRI ALT).

**Non-determinism by interaction**

- Synchronization on input channels only if channels are directed in Occam2.
- To initiate the sending of data (take 1 buffer [Top] - statement, then the process is stopped.
- Any Occam process can be employed in the ALT statement.
- The ALT statement is non-deterministic (there is also a deterministic version PRI ALT).

**Non-determinism in Occam2**

- Guards are referring to boolean expressions and channel input operations.
- The boolean expressions are local expressions, i.e., if none of them evaluates to true, then the process is stopped.
- If all triggered channel input operations evaluate to false, the process is suspended until subsequent activity on one of the channels.
- Any Occam process can be employed in the ALT statement.
- The ALT statement is non-deterministic (there is also a deterministic version PRI ALT).

**Non-determinism by interaction**

- Guards are referring to boolean expressions and channel input operations.
- The boolean expressions are local expressions, i.e., if none of them evaluates to true, then the process is stopped.
- If all triggered channel input operations evaluate to false, the process is suspended until subsequent activity on one of the channels.
- Any Occam process can be employed in the ALT statement.
- The ALT statement is non-deterministic (there is also a deterministic version PRI ALT).

**Non-determinism in Occam2**

- Guards are referring to boolean expressions and channel input operations.
- The boolean expressions are local expressions, i.e., if none of them evaluates to true, then the process is stopped.
- If all triggered channel input operations evaluate to false, the process is suspended until subsequent activity on one of the channels.
- Any Occam process can be employed in the ALT statement.
- The ALT statement is non-deterministic (there is also a deterministic version PRI ALT).
Non-determinism

Selective Synchronization

Message-based selective synchronization in Ada

Forms of selective waiting:

```
select_statement ::= selective_accept |
    conditional_entry_call |
    timed_entry_call |
    asynchronous_select
```

selective_accept :=

```
select |
[guard] selective_accept_alternative }
```

If none of the entries have waiting calls
or
if all conditions evaluate to true
or
if none of the open entries have waiting calls before the deadline specified by the underlying concept: Dijkstra's guarded commands.

```
else
    when <condition> =>
```

If exactly one of the entries has waiting calls
when
<condition> =>

```
```

This situation occurs if:

- all tasks which can possibly call on any of the open entries are terminated,
or
- all remaining tasks which can possibly call on any of the open entries are terminated,
or
- all remaining tasks which can possibly call on any of the open entries are terminated,
or
- all remaining tasks which can possibly call on any of the open entries are terminated,
or
- all remaining tasks which can possibly call on any of the open entries are terminated.

```
```

If some condition evaluate to true or the accept statement after those conditions is treated like in the previous form.

```
```

If conditions evaluate to false in program, it failed. Hence it is important that the set of conditions covers all possible states.

```
```

This is e.g. useful to probe the state of a server before committing to a potentially blocking call.

```
```

The underlying concept: Dijkstra's guarded commands.

```
```

If none of the currently open entries have no waiting calls or the deadline has expired.
There is only one of the currently open entries waiting calls or the deadline has expired.

```
```

If all conditions are true.
This situation occurs if:

- the task is terminated, i.e. the task is terminated.
```

```
```

When some of the open entries have waiting calls and none of them can be called again.
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Non-determinism

Selective Synchronization

Timed entry-calls

Timed entry-call:

```ada
select entry_call_statement [sequence_of_statements] 
or delay_alternative 
end select;
```

- If the call is not accepted before the deadline specified by the delay alternative or the idle alternatives chosen, this is e.g. useful to withdraw an entry call after some specified time-out.
- There is only one entry-call and one delay alternative.

Example:

```ada
select
  controller.Request [some_item]; 
  process data
or delay 45.6; 
end select;
```

Forms of selective waiting:

- `selective_accept`: 
- `conditional_entry_call`:
- `asynchronous_select`:

- The possibility to escape a running code block due to an event from outside this task.

... 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)

Correctness of non-deterministic programs

Partial correctness:

(P1) = terminates (Program(I,O)) \rightarrow Q(I,O)

Total correctness:

(P1) \rightarrow \exists Q. (Program(I,O) \rightarrow Q(I,O))

Safety properties:

(P1) \land Processes(I,S) \rightarrow Q(I,S)

where Q means that Q always holds.

Liveness properties:

(P1) \land Processes(I,S) \rightarrow Q(S) \land S(I,S)

where Q means that Q eventually holds and will then stay true and S is the current state of the concurrent system.

Non-determinism

Sources of Non-determinism

As concurrent entities are not in "lockstep" synchronization, they "overstep" each other and arrive at synchronization points in indeterministic order, due to (just a few):

- Operating systems / runtime environments:
  - Schedules are often non-deterministic.
  - System load affects an influence on concurrent execution.
- Message passing systems react load depended.
- Networks & communication systems:
  - Traffic arrival is 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.
- Physical systems and computer systems connected to the physical world are non-deterministic.

Non-determinism by interaction:

- When you formulate a non-deterministic statement like the one on the left you need to simulate an invariant which holds true whichever alternative will actually be chosen.
- This is very similar to finding loop invariants in sequential programs.

Non-Determinism

Non-determinism by design:

- Benefits & Limitations

Non-determinism by interaction:

- Selective synchronization
- Selective accepts
- Selective calls

Correctness of non-deterministic programs:

- Sources of non-determinism
- Predicates & invariants