Non-determinism

Definition

A property of a computation which may have more than one result.

Non-determinism by design:

Motivation for non-deterministic design

By explicitly leaving the sequence of evaluation or execution undetermined:

- The compiler / runtime environment can directly (i.e. without any analysis) translate the source code into an concurrent implementation
- The implementation gains potentially significantly in performance
- The programmer does not need to handle any of the details of a concurrent implementation (context loss, messages, synchronizations, ...)

A programming language which allows for those formulations is required!

Numerical non-determinism in concurrent statements (Chapel):

\[ \text{reduce} \left[ i \in 1..1000000 \right] i^2 \]

Depending on numeric type

Those formulations is required!

The programmer needs to understand the numerical implications of out-of-order expressions.

Non-determinism by interaction

Selective waiting in Occam 2

\[ \text{ALT} \]

\[ \text{Guard1: } \text{NumberInBuffer} < \text{Size} \] \text{append} \ ? \ \text{Buffer} \ [\text{Top}] \]

\[ \text{SEQ} \]

\[ \text{NumberInBuffer} := \text{NumberInBuffer} + 1 \]

\[ \text{Top} := (\text{Top} + 1) \mod \text{Size} \]

\[ \text{Guard2: } \text{NumberInBuffer} > \text{Size} \]

\[ \text{SEQ} \]

\[ \text{Take} \ ? \ \text{Buffer} \ [\text{Base}] \]

\[ \text{NumberInBuffer} := \text{NumberInBuffer} - 1 \]

\[ \text{Base} := (\text{Base} + 1) \mod \text{Size} \]

- Guards are referring to boolean expressions and/or channel input operations.
- The boolean expressions are local expressions, i.e. if none of them evaluates to true at the time of the evaluation of the ALT-statement, then the process is stopped.
- If the triggered channel input operations evaluate to false, the process is suspended until further activity on one of the matched channels.
- Any Occam 2 process can be employed in the ALT-statement.
- The ALT-statement is non-deterministic (there is also a deterministic version PRE ALT).
Non-determinism

Selective Synchronization

Message-based selective synchronization in Ada

Basic forms of selective synchronization

(select guarded-accept)

select

when <condition> => accept ...

or

when <condition> => accept ...

or

when <condition> => accept ...

end select;

• If all conditions are 'true'

• If some condition evaluates to 'true'

• If <condition> evaluates to 'false'

This form is identical to Dijkstra's guarded commands.

Basic forms of selective synchronization

(select guarded-accept-delay)

select

when <condition> => accept ...

or

when <condition> => accept ...

or

when <condition> => accept ...

end select;

• If all currently open entries have no waiting calls or all entries are closed

• The else alternative is chosen, the associated statements are executed, and the select statement complete

• Otherwise if one of the open entries with waiting calls is chosen as above

This form never suspends the task.

Selective Synchronization

Conditional entry-calls

select

conditional_entry_call |

timed_entry_call |

underlying concept: Dijkstra's guarded commands

conditional_entry_call and timed_entry_call implement...

... the possibility to withdraw an outgoing call...

... this might be restricted if calls have already been partly processed

© 2018 Uwe R. Zimmer, The Australian National University page 386 of 757 (chapter 4: “Non-determinism” up to page 394)
Non-determinism

Selective Synchronization

Timed entry-calls

timed_entry_call :=
  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 last alternative chosen.
- This is e.g. used to withdraw an entry call after some specified time-out.

Example

select
  controller.Request (Some_Item);
  ---- process data
  or
  delay 45 s; ---- seconds
  --------- try something else
  end select;

As concurrent entities are not in “lockstep” 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.
- Networks & communication systems:
  - Traffic will arrive in an unpredictable way (non-deterministic).
  - Example: call after some specified time-out.
- Computing hardware:
  - Timers drift and clocks have granularities.
- Physical systems (and computer systems connected to the physical world): are intrinsically non-deterministic.

Correctness of non-deterministic programs

Partial correctness:

\[ P(I) \land \text{terminates}\(\text{Program} \mid I, O\) \Rightarrow Q(I,O) \]

Total correctness:

\[ P() \Rightarrow (\text{terminates}\(\text{Program} \mid I, O\) \land Q(I,O)) \]

Safety properties:

\[ (P() \land \text{Processes} \mid I, S) \Rightarrow [Q() \land S] \]

where \( \equiv Q() \) means that \( Q() \) does always hold

Liveness properties:

\[ (P() \land \text{Processes} \mid I, S) \Rightarrow [Q() \land S] \]

where \( \equiv Q() \) means that \( Q() \) does eventually hold (and then stay true) and \( S() \) is the current state of the concurrent system

Correctness of non-deterministic programs

- Correctness predicates need to hold true irrespective of the actual sequence of interaction points.
- Correctness predicates need to hold true for all possible sequences of interaction points.

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.

Summary

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