Concurrent Systems

3D4 ⟷ CS2016

Operating Systems

Andrew Butterfield
ORI.G39, Andrew.Butterfield@scss.tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

with thanks to Mike Brady

Promela Verification Constructs

- Basic Assertions
- End-state Labels
- Progress-State Labels
- Accept-State Labels
- Never Claims
- Trace Assertions

Promela – *Basic Assertion*

- Format:

```
assert(<expression>)
```

where `<expression>` must evaluate to `true` or

  - execution will be aborted or
  - verification will fail

Note: an `assert` is evaluated during execution and verification.

Promela – *End-State Label*

- Promela always verifies that no deadlock occurs.
- It assumes that the only valid end-states for a system are where
  - each process is the end of its code.
- If it can show an end state where a process is not at the end of its code, it considers that an error.
- Sometimes, it is legitimate for a process not to end up at the end of its code. You can label such states **`end`**... to indicate to Promela that they are valid (non-erroneous) end points.

  `end`, **`end_one`**, **`end00`** – anything starting with **`end`**.

Promela – *Progress-State Label*

- A system can have loops – cycles of sequences of states passed through infinitely often.
  - The question is, are such loops desirable or not;
    - if progress is made each time, then they are desirable loops -- progress cycles;
    - otherwise, they are undesirable *non-progress cycles*, where the system is doing something but not progressing.

Promela – *Progress-State Label (2)*

- We can label a state to be a progress state using a **`progress`**... label (same idea as the **`end`**... label).
- We can check that every potentially infinite cycle passes through a progress state.
- If not, we have non-progress cycles, which can lead to starvation elsewhere.

Promela – *Accept-State*

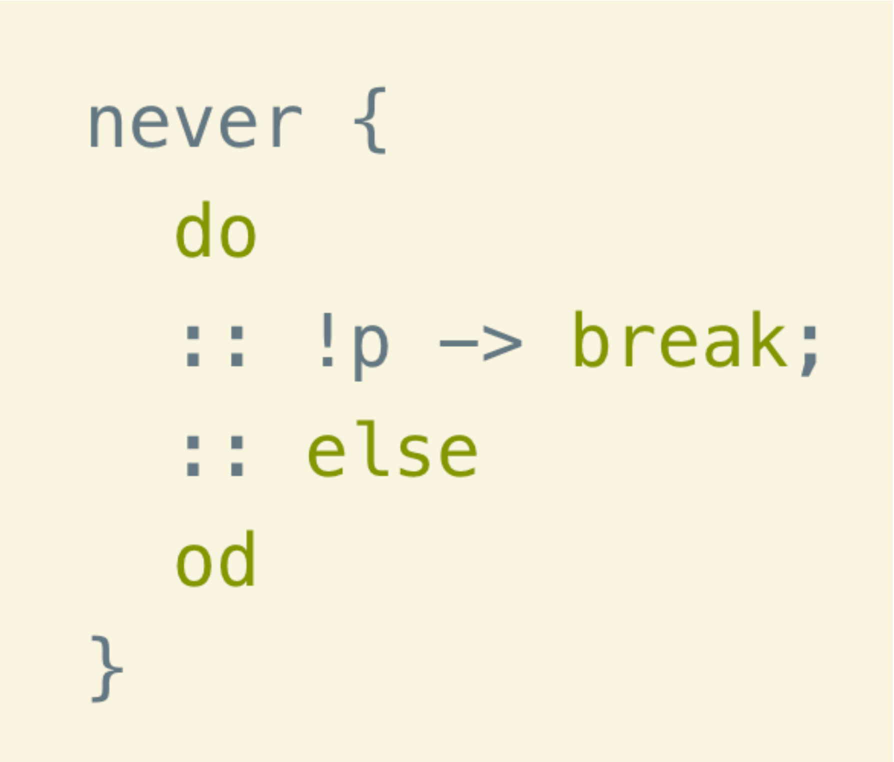
- We can label a state to be a accept state using a **`accept`**… label (same idea as the **`end`**… label).
- We can check any accept state in a cycle is not executed infinitely often
- There are usually generated automatically with the use of Never claims

Promela – *Never Claims*

- All of the verification checks so far focus on individual, often labelled, execution states.
- Also required are properties about sequences of states that arise during execution.
  - This requires performing checks at every step of an execution, not just at designated states.
- A never claim is written as **`never { <proc> }`** where **`<proc>`** is a process that describes a behaviour that should NOT happen
  - This undesired behaviour is considered to have "happened" if **`<proc>`** terminates.
- The verifier will check the possible system behaviours against the behaviour described in the never claim.

Invariant checking using a never claim

- We want to assert that property **`p`** is true all the time
    - One way is to add **`assert{p}`** after every statement in the model
        - Awkward
- Another way is to add a never claim
  with a process that loops as long as the assertion is true,
  but which immediately exits if not.
- Never claims are quite hard to write,
  but often it is possible to write a description
  of correct behaviour using temporal logic
  and let SPIN generate the never claim.



```
never {
  do
  :: !p -> break;
  :: else
  od
}
```