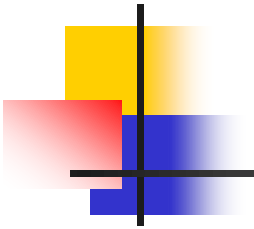


Properties of concurrent systems in Promela

`assert` *statement*

- defines local invariants
  - properties that must be executed at specific points in the program
- **`assert(expr)`**
- **`expr`** = true
  - the **`assert`** statement has no effect.
- **`expr`** = false
  - during the simulation, SPIN will display the message “Error: assertion violated”.
  - during verification, violation of **`assert`** statements is checked on all final calculations.
- In *simulation mode*, only system properties described by the **`assert`** operator can be checked.
  - the other kinds of property specifications can be checked only in *verification mode*.

Properties of concurrent systems in Promela

```
#define p 0
#define v 1
chan sem = [0] of { bit };
proctype dijkstra(){
            byte count = 1;
  endpoint: do :: (count == 1) -> sem ! p; count = 0
               :: (count == 0) -> sem ? v; count = 1
            od }
active[3] proctype user(){
          sem ? p;
          CritSection
          sem ! v;
          NonCritSection
```

End-state labels `end*`

- Marks normal final states for the verifier
  - Presence of deadlocks
- Normal final state:
  - some processes have reached the end of their program body, and
  - all message channels are empty.
    - not *every* process reaches the end of its body.

Properties of concurrent systems in Promela

Active-state labels **`progress*`**

- marks operators whose execution is desired
  - progress in model behavior (liveness property)
    - during verification, SPIN checks that any infinite loop passes through at least one **`progress*`** label
      - in case of violation, SPIN reports the existence of a non-progressive cycle (possible starvation of processes)

```
proctype dijkstra(){
            byte count = 1;
  endpnt: do :: (count == 1) ->
  progresspnt:              sem ! p; count = 0
              :: (count == 0) ->
                            sem ? v; count = 1
            od }
```

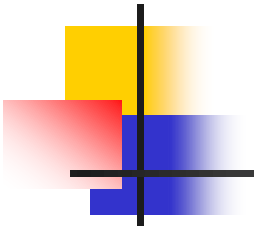
- Progress: the semaphore is necessary for processes infinitely
  - during verification, SPIN will check that in any infinite computation, the semaphore is used an infinite number of times.

Properties of concurrent systems in Promela

Accept-state labels `accept*`

- labels undesired conditions
  - "Undesired" trajectories pass an infinite number of times accepting state
  - during verification, SPIN checks that there are *no calculations* in the system that pass through the operators labeled with `accept*` infinitely often.

```
proctype dijkstra(){
            byte count = 1;
  endpnt: do :: (count == 1) ->
  progresspnt:              sem ! p; count = 0
              :: (count == 0) ->
  acceptpnt:                sem ? v; count = 1
            od }
```



Properties of concurrent systems in Promela

Process **`never`**

- makes it possible to define global invariants
  - **`assert`**, **`end`**, **`accept`**, **`progress`** are not checked in all system states.
- is a description of the behavior that should not occur in the system.
- is used to monitor system behavior:
  - does not affect the state
    - cannot declare variables
    - cannot change the value of a variable
    - cannot manipulate message channels
- in a model, there can only be single process **`never`**.
- is considered only during verification.
- allows you to check the system property
  - in initial condition and after each calculation step
    - after each statement of any process

Properties of concurrent systems in Promela

- Checking property **p** at each step of the system:

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

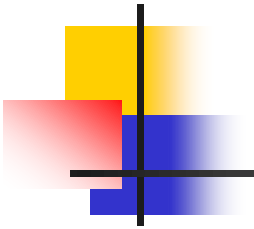
- It is executed at every step of the system.
- If condition **p** is false
  - the **never** process is aborted, passing to the final state.
    - Termination of **never** is interpreted as erroneous behavior of the system.
- If **p** is always true,
  - the **never** process remains in the loop
    - there is no error in the system.

Properties of concurrent systems in Promela

- Checking property **p** at each step of the system:
  - without **never**

```
active proctype monitor(){
        atomic { !p -> assert(false) }
}
```

- The **monitor** process can initiate the execution of an **atomic** block at any point in the computation of the system.
  - In any reachable state of the system in which the invariant **p** is violated,
    - **monitor** reports an error using the **assert** statement.



Properties of concurrent systems in Promela

- *Always, if* p *has become true, then in the future* q *will become true, and* p *will remain true until* q *becomes true.*
  - LTL: **G**(p → (p **U** q))
- In model checking, we are not interested in computations in which the property holds
  - for the model, we try to find computations in which the property is violated
    - **!G**(p → (p **U** q))
    - *p* became true, and *q* is false forever, or *p* became false before *q* became true.

Properties of concurrent systems in Promela

- Violation of a property with *q* remains false always
  - only on infinite computations
    - `assert` will not work
      - checked only on finite computations
- In verification mode, in the initial state of the system,
  - SPIN checks the possibility to execute the first operator of the never process.
  - label `S0`: non-deterministic cycle.

```
never { /* !G (p → (p U q)) */
S0:       do
             :: p && !q -> break
             :: true
          od;
accept:   do
             :: p && !q
             :: !(p || q) -> break
          od
}
```

Properties of concurrent systems in Promela

- Condition `true`
  - always executable and does not affect the computation.
  - returns the `never` process to its initial state.
  - keeps a progress

```
never { /* !G (p → (p U q)) */
S0:       do
             :: p && !q -> break
             :: true
          od;
accept:   do
             :: p && !q
             :: !(p || q) -> break
          od
}
```

- Condition `p&&!q`
  - model behavior: *p* has become true, but *q* is not true yet.
  - from this state, an incorrect path of the analyzed program may begin.

Properties of concurrent systems in Promela

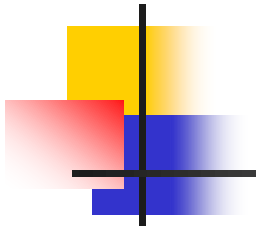
- Incorrect path
  - in all subsequent states *p* will be true and *q* will be false
    - infinite stay in label `accept`
  - there is a state in which neither *p* nor *q* will be true
    - termination of the `never` process

```
never { /* !G (p → (p U q)) */
S0:       do
             :: p && !q -> break
             :: true
          od;
accept:   do
             :: p && !q
             :: !(p || q) -> break
          od
}
```

Properties of concurrent systems in Promela

- *p* and *q* are true
  - none of the selection conditions is satisfiable
    - the `never` process is blocked.
      - blocking is desired behavior
        - is not terminated and
        - does not pass an infinite loop labeled `accept`

```
never { /* !G (p → (p U q)) */
S0:       do
             :: p && !q -> break
             :: true
          od;
accept:   do
             :: p && !q
             :: !(p || q) -> break
          od
}
```



Properties of concurrent systems in Promela

- Linear Temporal Logic formulas
  - temporal operators **F** is **<>**, **G** is **[]**, **U** and **W**
  - atomic propositions are Promela Boolean expressions with names starting with a lower-case letter
  - Boolean operators **`&&`**, **`||`**, **`->`** and **!**

  - `ltl prp {[](p -> (p U q))}`