Computer Aided Verification 2015
The SPIN model checker

Grigory Fedyukovich
<grigory.fedyukovich@usi.ch>

Universita' della Svizzera Italiana

March 30, 2015

Material borrowed from Roberto Bruttomesso

Outline

Outline

Spin

Two modes

- Simulator (done): it runs a guided or a random simulation of the model defined by the user. Can be used to quickly check the behavior of a model
- Verifier (today): it generates an executable whose run **exhaustively** checks the validity of the property (or the absence of deadlocks)
- Notice the difference between the two, one trace as opposed to all traces

Outline

Verifier

- Precisely, it is a C program derived from the promela model, that can be compiled into an executable that explores the state space
- It returns some information, such as
  - reachable and unreachable states
  - invalid end states (deadlock)
- From the command line:
  - `spin -a filename`
  - a C program `pan.c` appears in the current directory
  - `gcc -o pan pan.c` generates and executable `pan`
  - `./pan` runs the verifier

Example

simple.pml

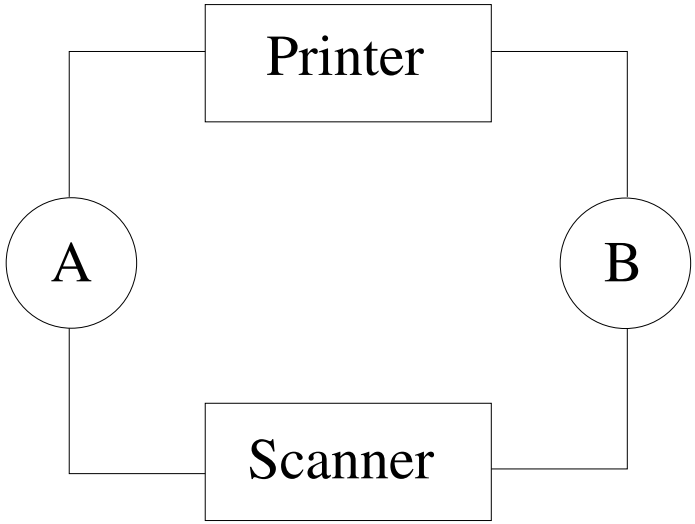
- spin -a simple.pml
- gcc -o pan pan.c
- ./pan

```
Full statespace search for:
    never claim                 - (none specified)
    assertion violations        +
    acceptance cycles           - (not selected)
    invalid end statespace      +
...
State-vector 44 byte, depth reached 3, errors:  0
unreached in proctype sender
line 11, state 6, "-end-"
    (1 of 6 states)
...
```

Discovering a deadlock

Two processes and two resources

File: `deadlock.pml`



Build and run the verifier

- `spin -a deadlock.pml`: writes `pan.c`
- `gcc -o pan pan.c`: compiles the verifier
- `./pan`: runs the verifier (writes `deadlock.pml.trail`)
- `./pan -r`: prints the trace that leads to the deadlock

Outline

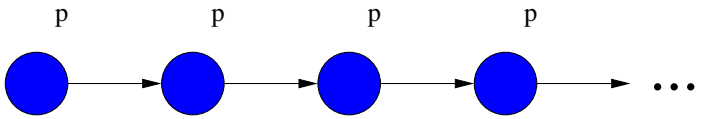
LTL properties specification

- In PROMELA it is possible to specify properties in LTL (Linear Temporal Logic)
- Safety properties:
  - Something bad never happens
  - Property of states
  - Can be computed with reachability methods (if the bad states are not reachable, the property holds)
- Liveness properties:
  - Something desirable eventually happens
  - Property of paths
  - More complex methods (ex. find a lazo-shaped path whose states violates the property)

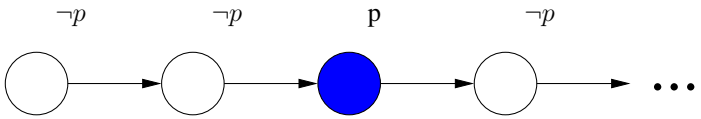
Modal Operators

And their PROMELA equivalent

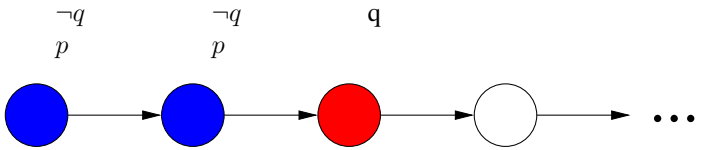
- Globally: property holds in every state
  Denoted with "G p" or "`[]` `p`"



- Finally: property eventually holds
  Denoted with "F p" or "`<>` `p`"



- Until: q never holds or p should hold until q holds
  Denoted with "p U q"

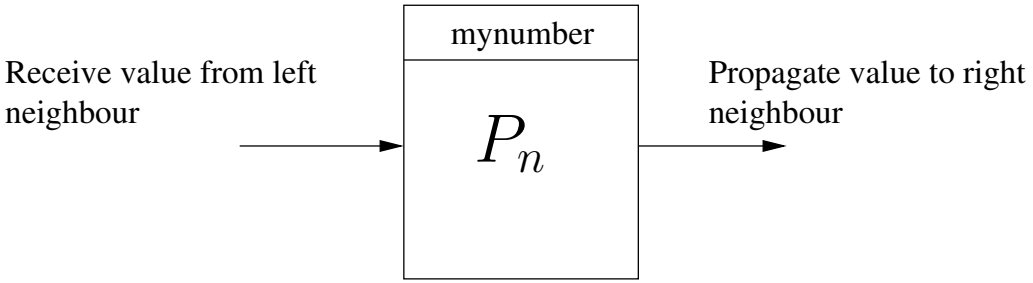


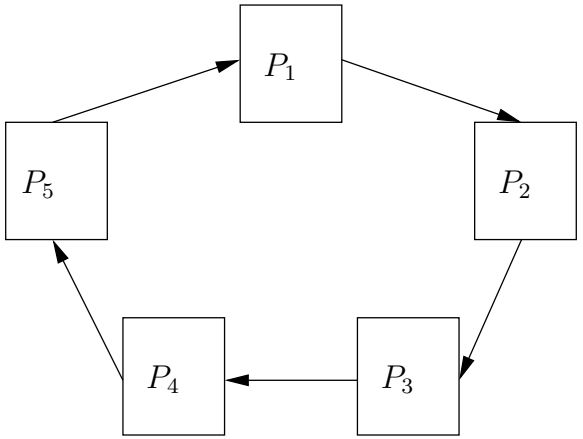
- Other useful operators: negation `!`, and `&&`, or `||`

LTL specification and checking

Leader election protocol

File: `leader.pml`





LTL specification and checking

Leader Election Protocol

File: `leader.pml`
Exercise: try the following properties

- `<>[]oneLeader`
- `![]noLeader`
- `<>elected`
- `[](noLeader U oneLeader)`

generate the corresponding PROMELA code with

- Syntax example: `ltl p1 { <> (nr_leaders == 0) }`
- Verify: `iSpin -> Verification -> use claim -> Run`
- Print counter-example (from the `*.trail` file): `iSpin -> Simulate/Replay -> (Re)Run`

LTL specification and checking

Semaphore

File: `dijkstra.pml`
Exercise: define the following properties, test them, and indicate if safety/liveness

- it is always the case that at most one user is in the critical section:

- it is always the case that eventually user 1 enters the critical region:

LTL specification and checking

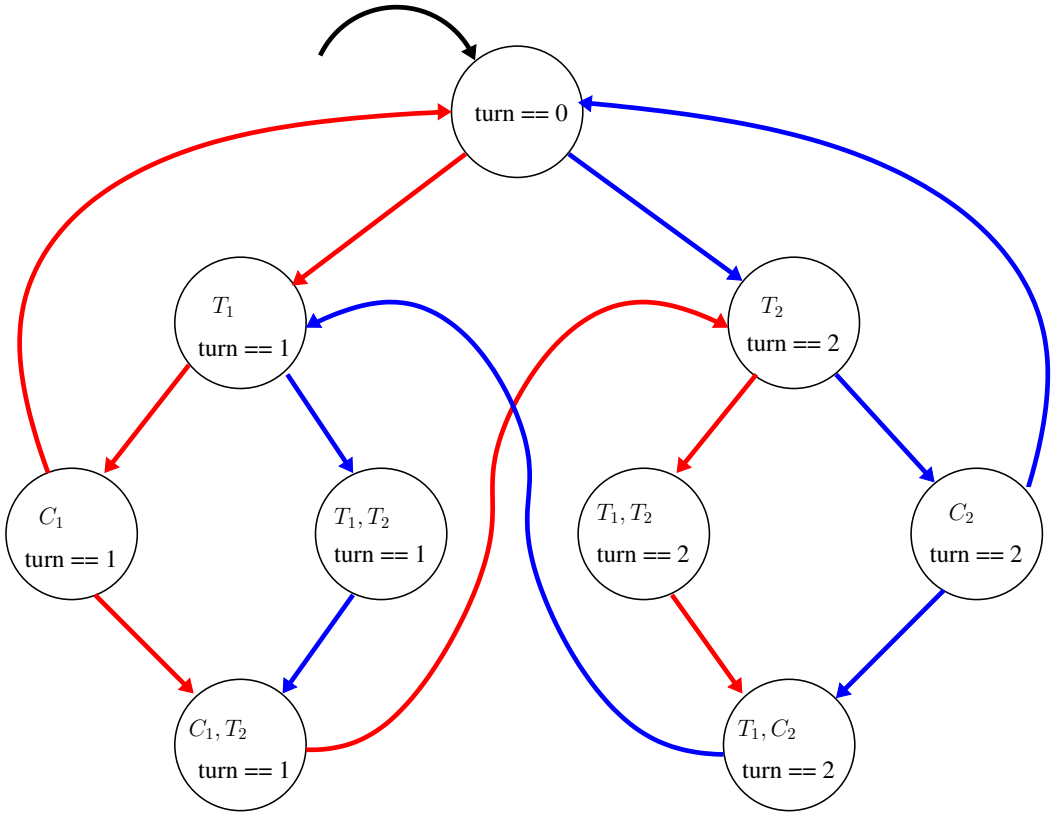
Semaphore

File: `dijkstra.pml`
Exercise: define the following properties, test them, and indicate if safety/liveness

- it is always the case that at most one user is in the critical section:
  - **G** (nr_users_in_section ≤ 1)
  - it is a safety property

- it is always the case that eventually user 1 enters the critical region:
  - **GF** (flags[1])
  - it is a liveness property

Mutual exclusion



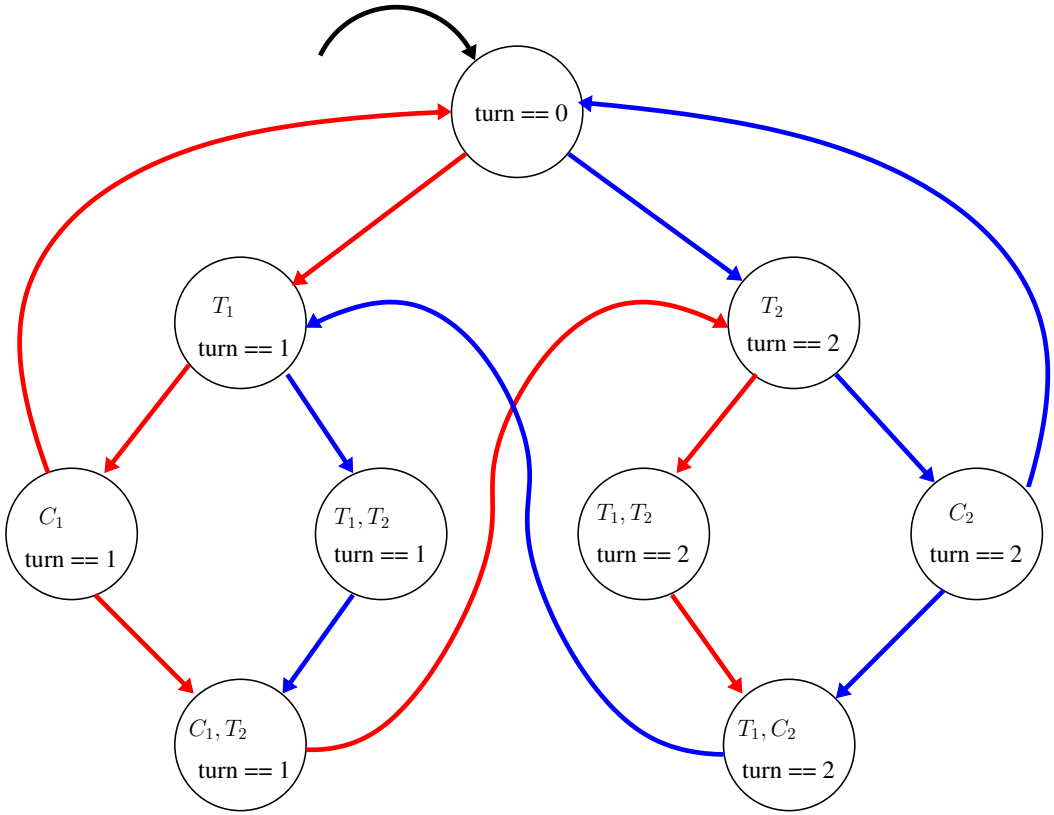
For $i = 1, 2$, $turn$:

- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

Mutual exclusion



For $i = 1, 2$, $turn$:

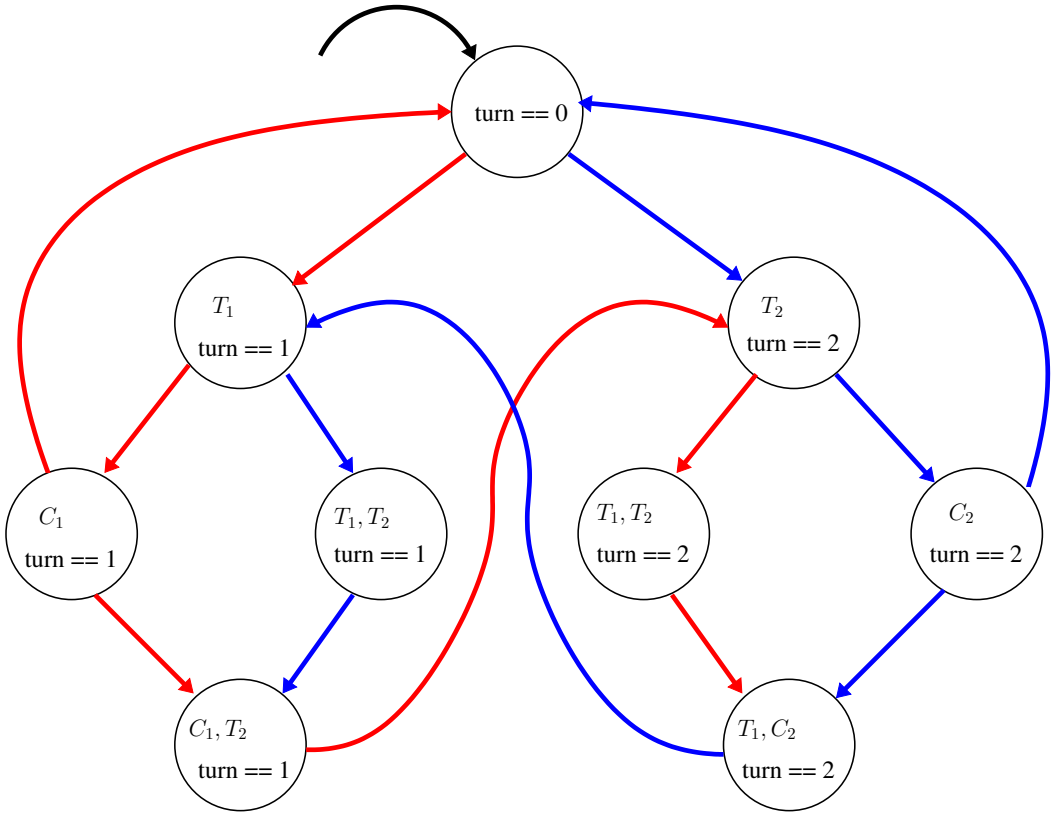
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{G}\neg(C_1 \wedge C_2) \ ?$$

Mutual exclusion



For $i = 1, 2$, $turn$:

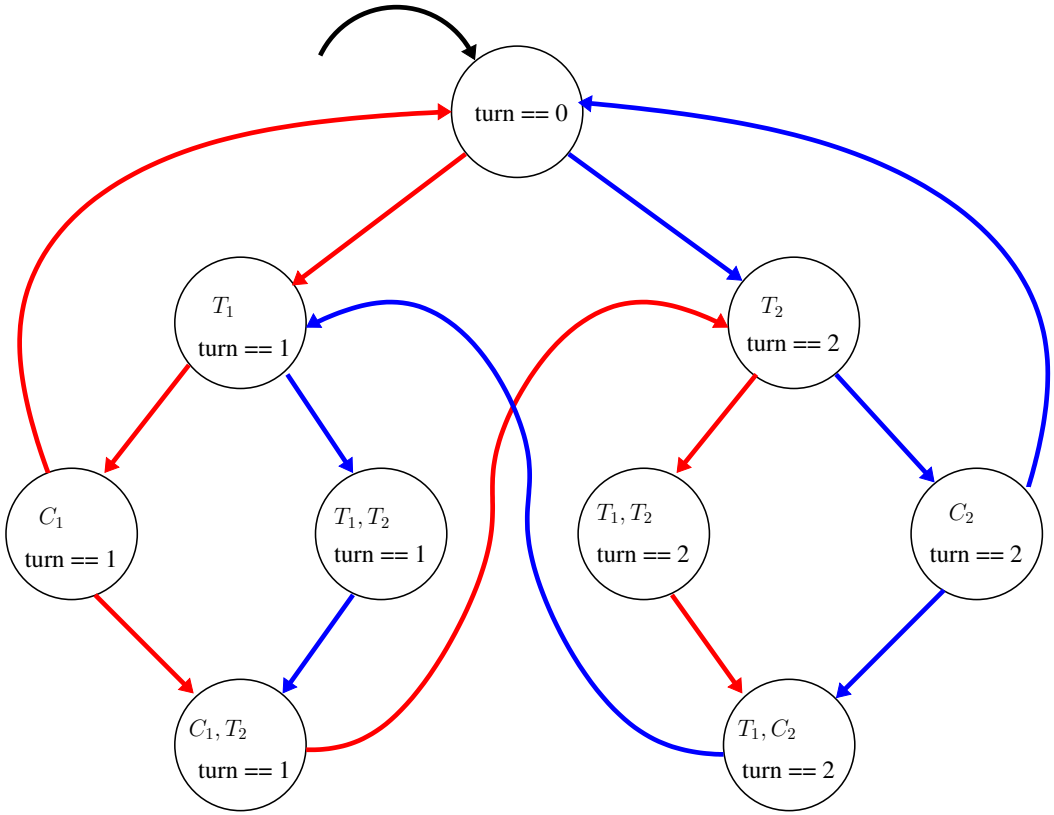
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{G}\neg(C_1 \wedge C_2) \text{ ? YES}$$

Mutual exclusion



For $i = 1, 2$, $turn$:

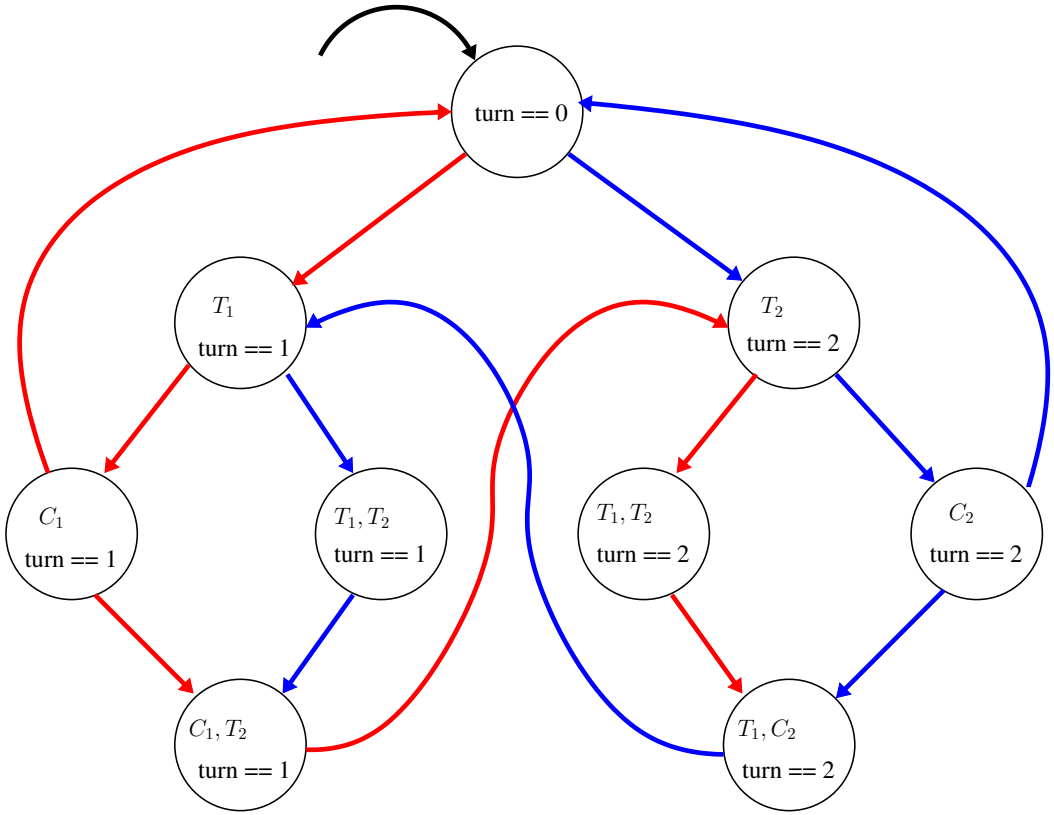
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{F} C_1 \ ?$$

Mutual exclusion



For $i = 1, 2$, $turn$:

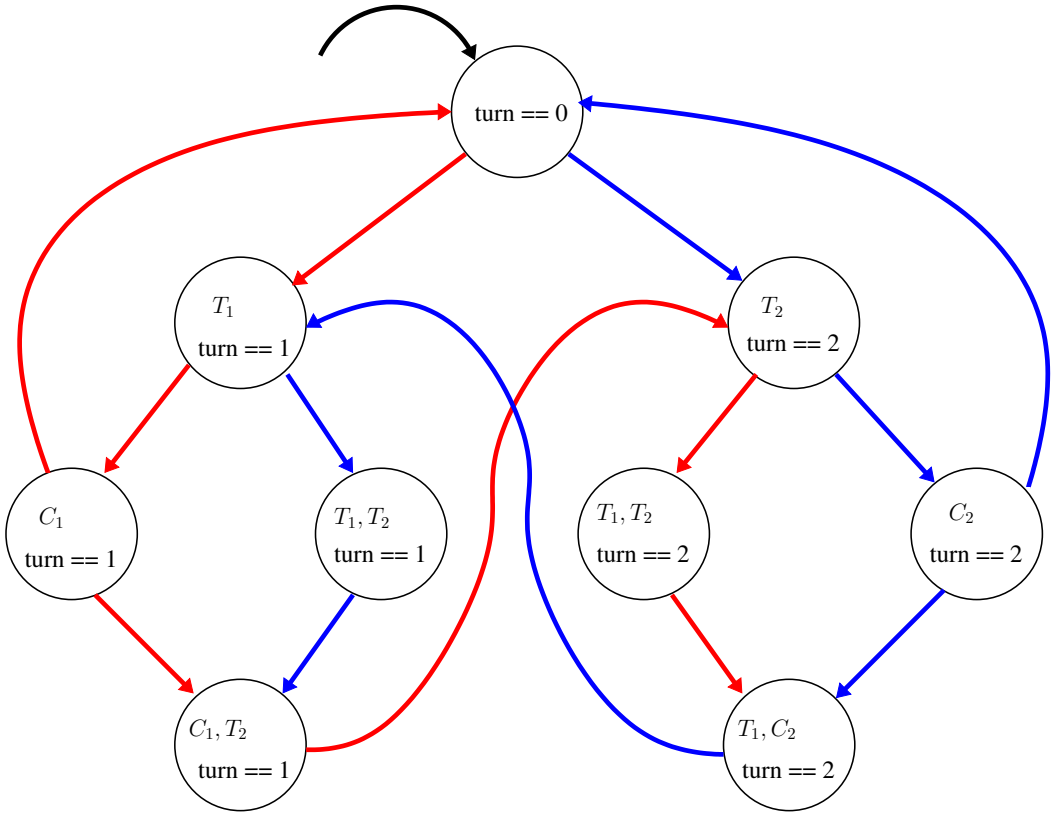
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$\mathbf{F}C_1$? NO

Mutual exclusion



For $i = 1, 2$, $turn$:

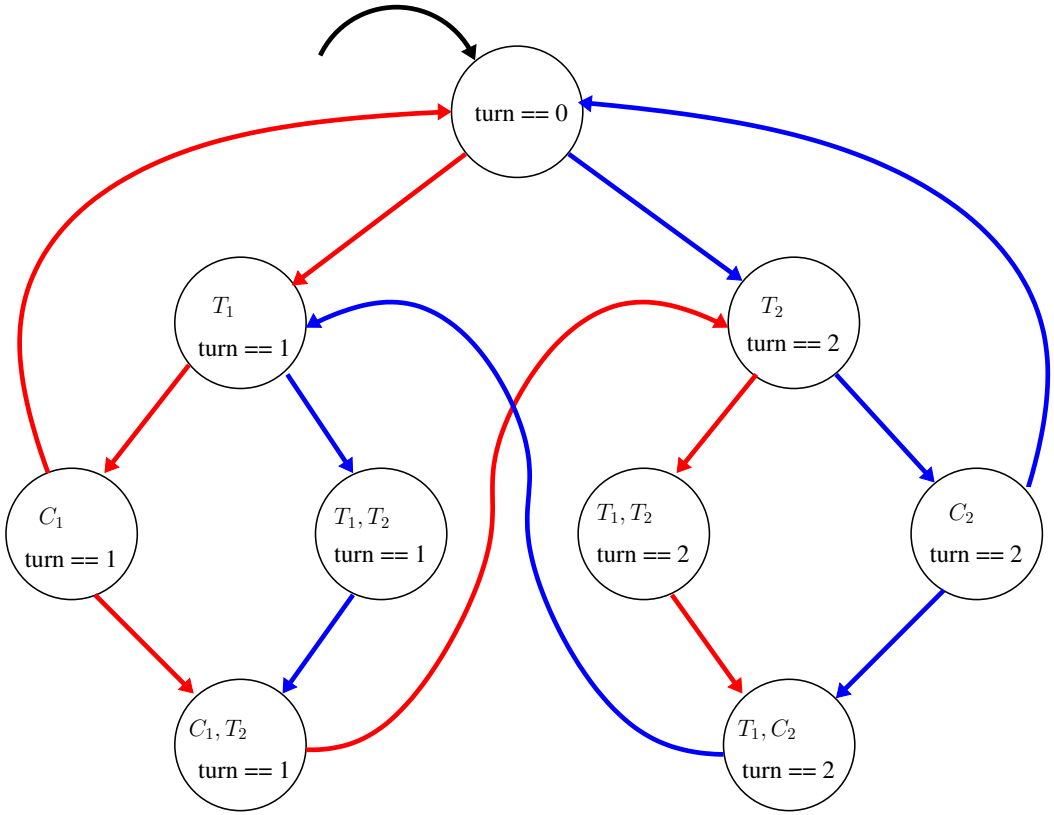
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{G}(T_1 \rightarrow \mathbf{F}C_1) \ ?$$

Mutual exclusion



For $i = 1, 2$, $turn$:

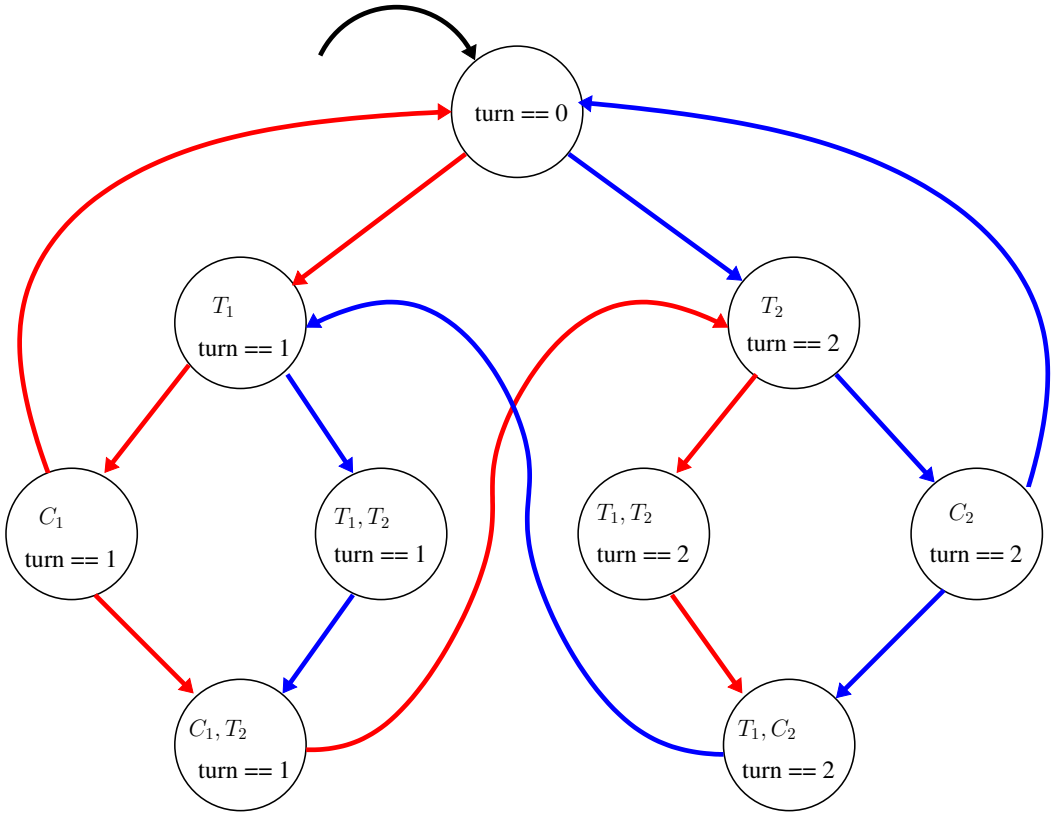
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{G}(T_1 \rightarrow \mathbf{F}C_1) \text{ ? YES}$$

Mutual exclusion



For $i = 1, 2$, $turn$:

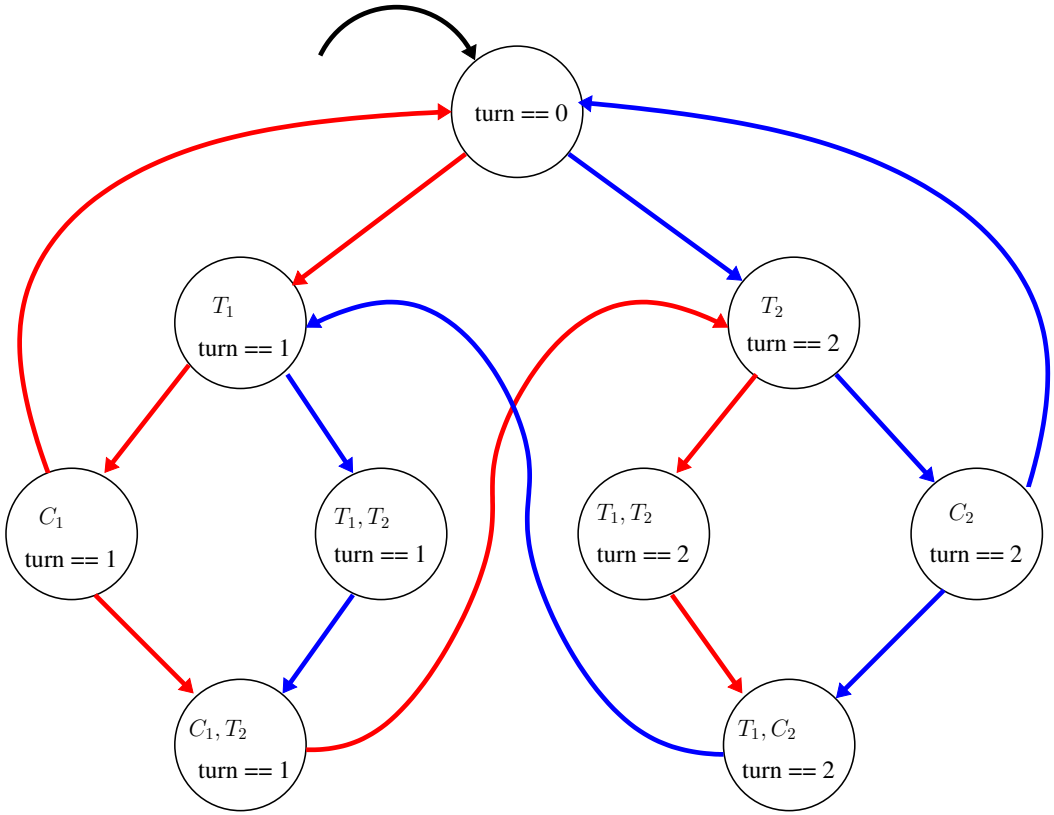
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{GF}C_1 \ ?$$

Mutual exclusion



For $i = 1, 2$, $turn$:

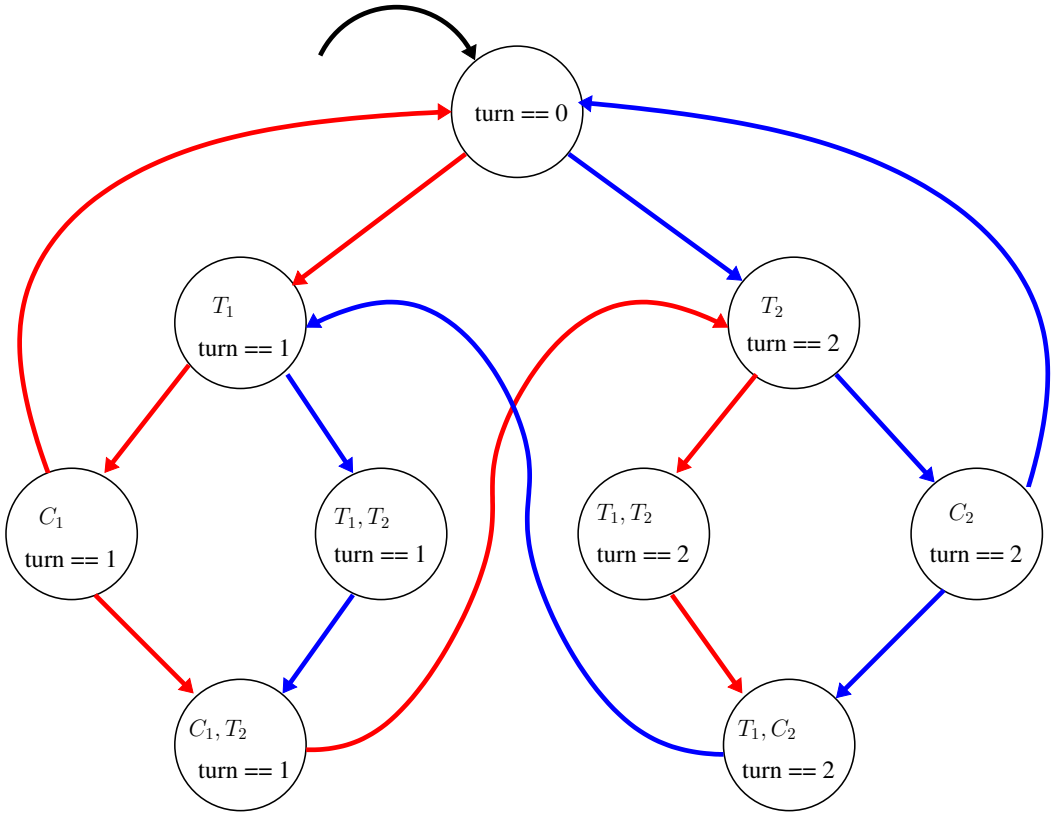
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$\mathbf{GF}C_1$? NO

Mutual exclusion



For $i = 1, 2$, $turn$:

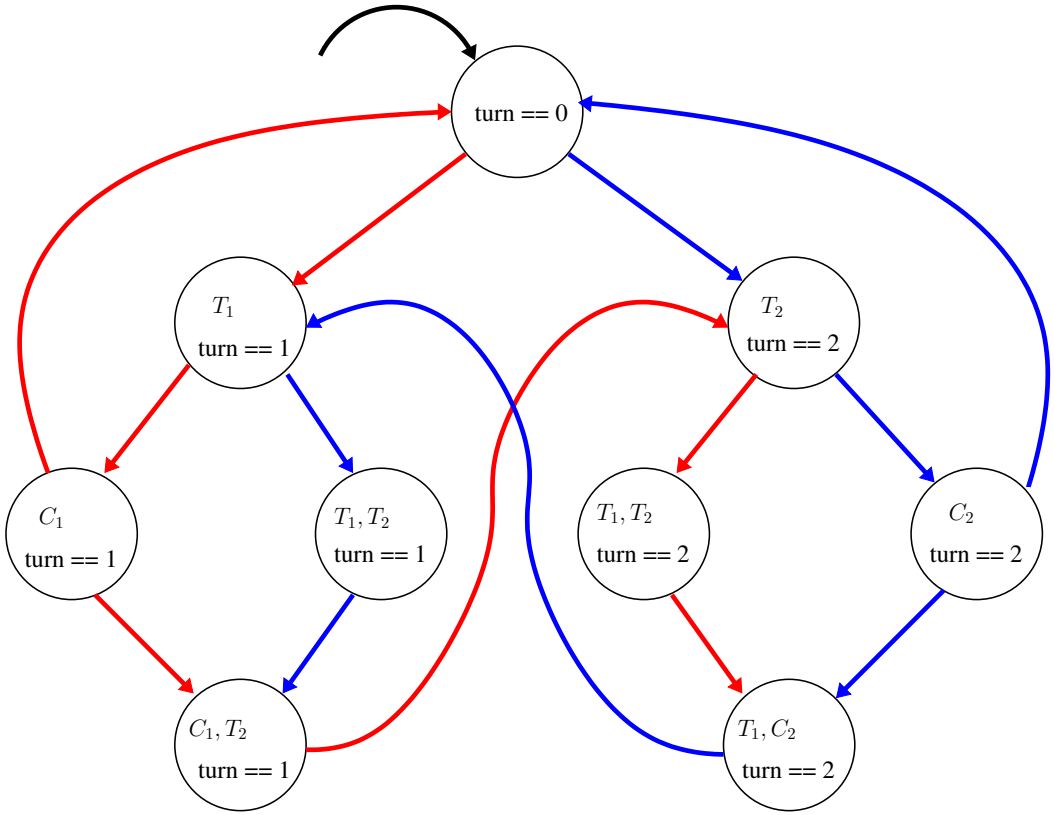
- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{GF}T_1 \rightarrow \mathbf{GF}C_1 \ ?$$

Mutual exclusion



For $i = 1, 2$, $turn$:

- $turn == i$: process i acquired access to critical region

T_i, C_i are boolean variables:

- T_i: process i wants to access the critical region
- C_i: process i is in the critical region

$$\mathbf{GF}T_1 \rightarrow \mathbf{GF}C_1 \text{ ? YES}$$