Information Systems Analysis

Temporal Logic and Timed Automata

(6)

System model verification in UPPAAL

© Paweł Głuchowski, *Wrocław University of Technology*

version 2.2

Contents of the lecture

Modelling patterns

- Controlling a transition by a clock
- Synchronised message passing between automata
- Multicast synchronisation with a specified number of receivers
- Urgent transition
- Timer
- Good practices

Contents of the lecture

Examples

- Mosquito
- Three escapees
- Petterson's algorithm
- Access to a file
- Trains and a gate

Modelling patterns

- Controlling a transition by a clock
- Synchronised message passing between automata
- Multicast synchronisation with a specified number of receivers
- Urgent transition
- Timer
- Good practices

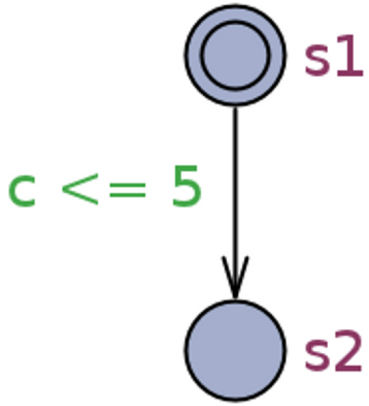
Modelling patterns

Controlling a transition by a clock

Automata A and B are controlled by a clock *c*. What differs them?

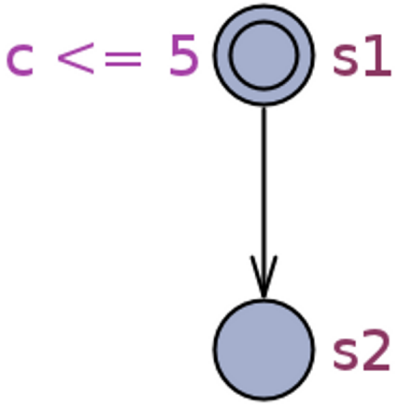
A: the clock controls the transition between states s1 and s2:

- the transition s1→s2 is <u>available</u> until 5 time units passes, but it may never occur:



E<> A.s1 and c > 5	YES
A<> A.s2	NO
E<> A.s2	YES

B: the clock controls the state s1 (as an *invariant*):



- the state s1 <u>must</u> be left until 5 time units passes:

E<> B.s1 and c > 5	NO
A<> B.s2	YES

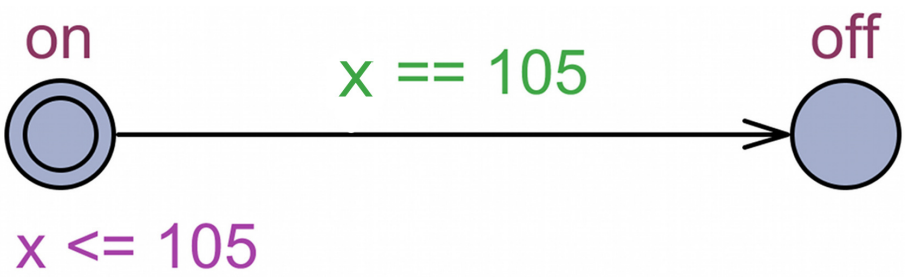
Modelling patterns

Controlling a transition by a clock

The automaton *Lecture* is controlled by a clock variable x.

- The lecture lasts (state *on*) exactly 105 minutes:

A<> Lecture.off	YES
A[] Lecture.on imply x <= 105	YES
A[] Lecture.off imply x >= 105	YES
E<> Lecture.on and x==105	YES
E<> Lecture.off and x==105	YES
Lecture.on --> Lecture.off	YES



Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

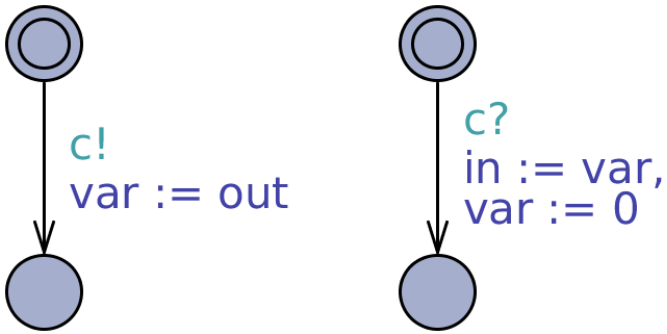
- Idea 1:
  - Synchronisation of automata is done through channels.
  - Data is passed through global variables.
- There are 4 variants available:
  - 1–way unconditional synchronisation,
  - 1–way conditional synchronisation,
  - 2–way unconditional synchronisation,
  - 2–way conditional synchronisation.

Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

- 1–way unconditional synchronisation:
  - the local variable ***out*** (left automaton) is passed to the local variable ***in*** (right automaton) through the global variable ***var,***
  - the passing takes places on synchronisation by the channel ***c***.

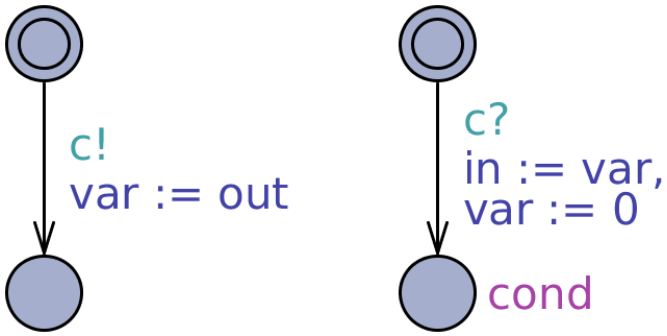


Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

- 1–way conditional synchronisation:
  - the local variable ***out*** (left automaton) is passed to the local variable ***in*** (right automaton) through the global variable ***var,***
  - the passing takes place on synchronisation by the channel ***c***,
  - the synchronisation takes places if the condition (invariant) ***cond*** is satisfied.

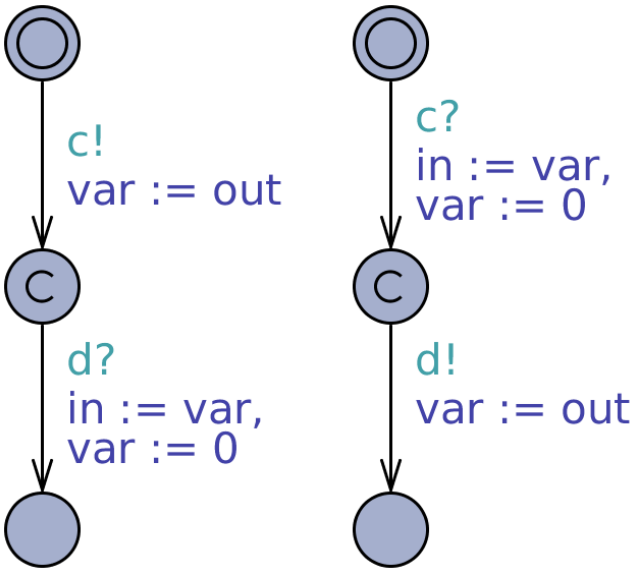


Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

- 2–way unconditional synchronisation:
  - the local variable ***out*** (left automaton) is passed to the local variable ***in*** (right automaton) through the global variable ***var,***
  - the passing takes place on synchronisation by the channel ***c***.
- At the same time:
  - the local variable ***out*** (right automaton) is passed to the local variable ***in*** (left automaton) through the global variable ***var,***
  - the passing takes place on synchronisation by the channel ***d***.

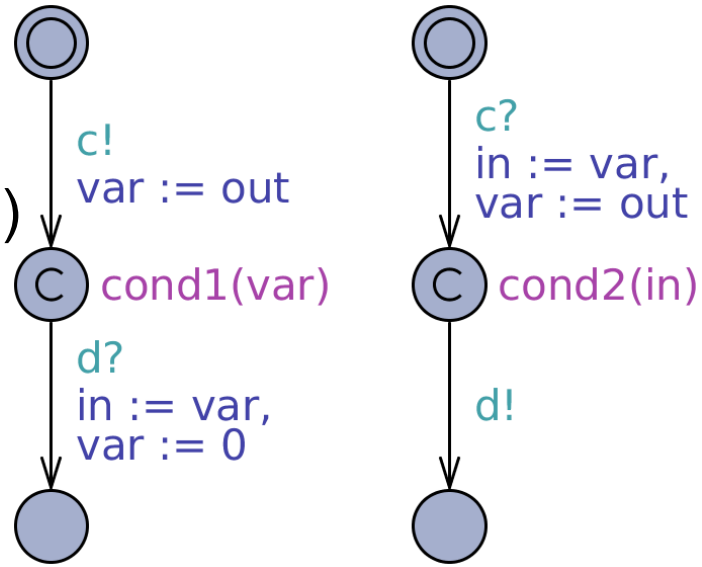


Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

- 2–way conditional synchronisation:
  - the local variable ***out*** (left automaton) is passed to the local variable ***in*** (right automaton) through the global variable ***var;***
  - the passing takes place on synchronisation by the channel ***c***, under the condition ***cond1*** (the right automaton receives a correct value).
- At the same time:
  - the local variable ***out*** (right automaton) is passed to the local variable ***in*** (left automaton) through the global variable ***var;***
  - the passing takes place on synchronisation by the channel ***d***, under the condition ***cond2*** (the left automaton receives a correct value).



Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

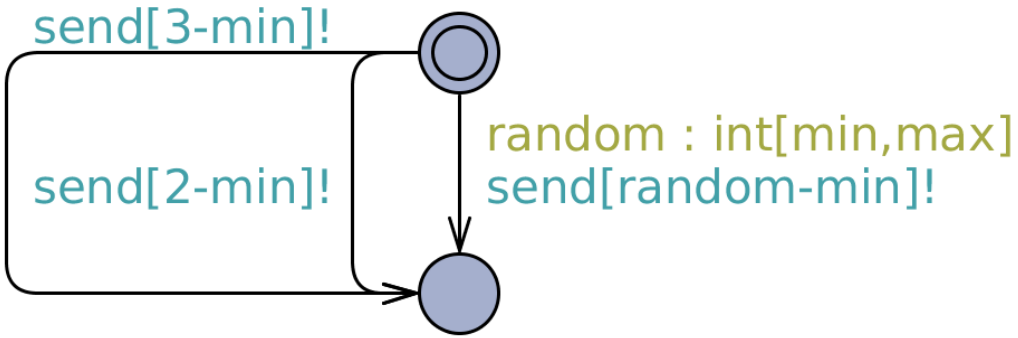
- Idea 2:
  - Synchronisation of automata is done through a table of channels.
  - A small integer value is passed through an adequate channel (its number is the value).

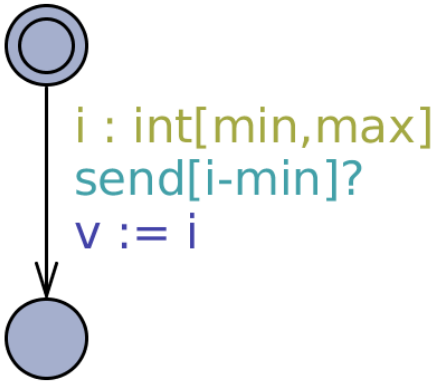
Modelling patterns

Synchronised message passing between automata

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 28—32)

- A number of type int and of the range [min,max] is passed.
- The synchronisation is done through a table of channels:
  chan send[max-min+1];
- The sender sends a number: 2, 3 or a randomly chosen one (*random*).
- The variable v receives the passed value.





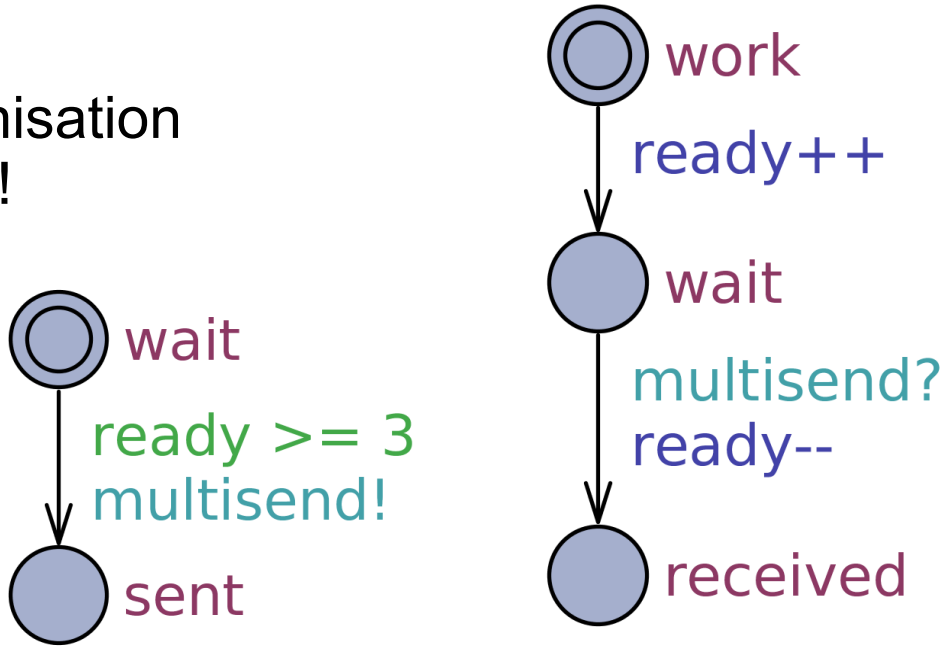
Verification for a large table of channels may take long time.

Modelling patterns

Multicast synchronisation with a specified number of receivers

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 32—33)

- Example: a synchronisation with at least 3 automata through a broadcast channel (*broadcast chan*) *multisend*:
  - the synchronisation takes place not before at least 3 receivers await it (*ready* >= 3).
  - without this condition the synchronisation could take place without receivers!

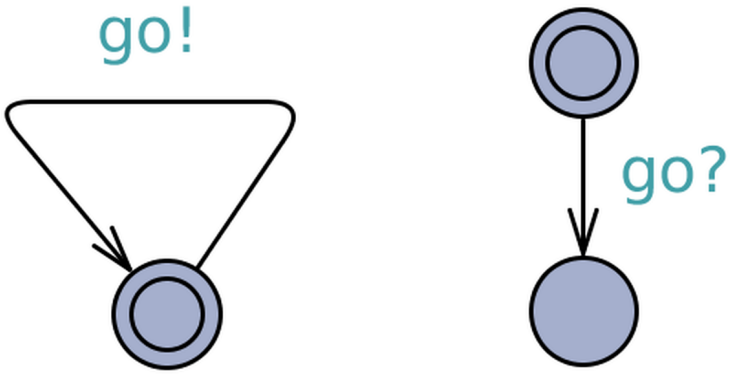


Modelling patterns

Urgent transition

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 34—35)

- **Assumption:** the outgoing state of the transition is not of a type *urgent* or *committed*.
- **Goal:** the transition will take place immediately, when its outgoing state becomes active.
- The transition is initialised by a channel *urgent chan* by a special automaton.



Modelling patterns

Timer

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 36—37)

- Timer allows e.g.:
  - to limit length of time of activeness of a given state or a group of states,
  - to designate a relative moment of synchronisation of automata.
- The flow of time is counted down from a given value (variable or constant) to zero.
- The timer is activated at a chosen moment.
- Reaching zero may cause certain reaction of an automaton (e.g. by synchronisation).

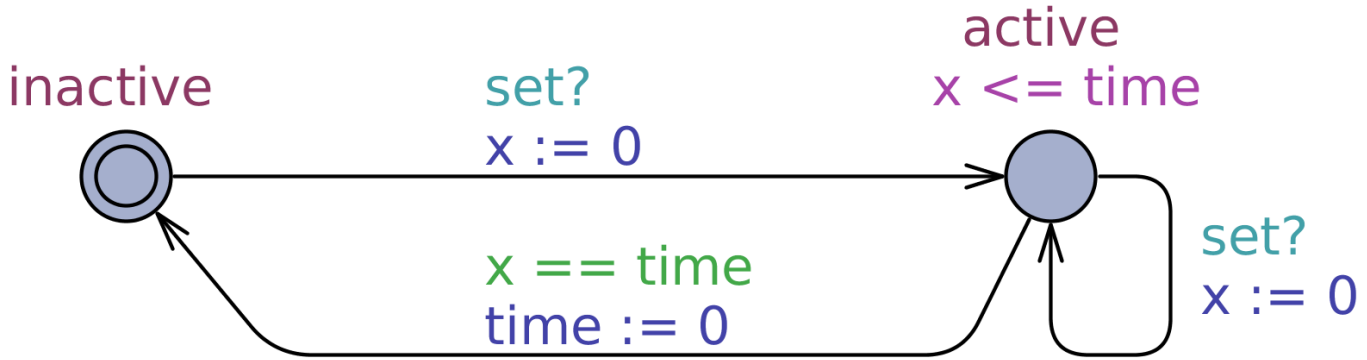
Modelling patterns

Timer

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 36—37)

A timer with a variable time of activeness:

- The timer is activated by the channel (*chan*) *set*.
- At the moment of activation of the timer the clock *x* is set to zero.
- The <u>variable</u> (*int*) *time* designates length of timer's activity time.
- At the moment of deactivation of the timer the variable *time* is set to zero.
- Checking if the counted time has passed: *time* == *0*.
- It is possible to reset the timer during its work.



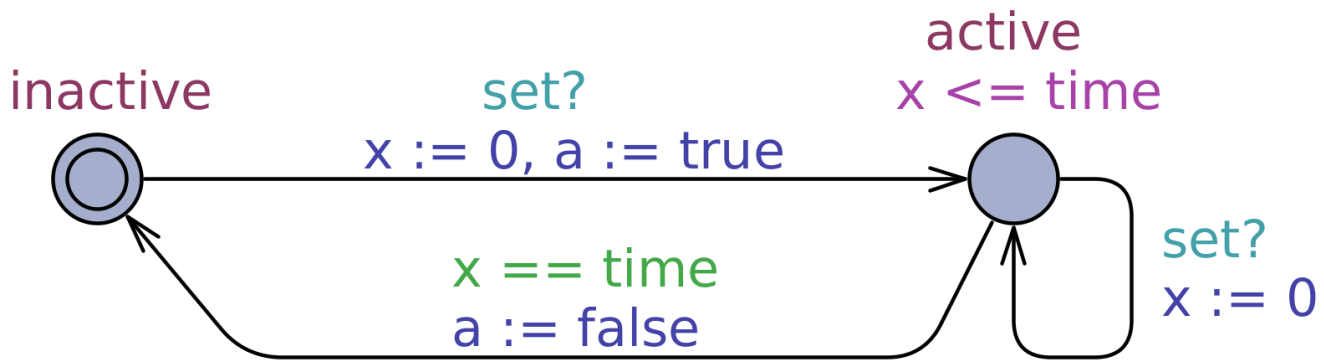
Modelling patterns

Timer

(based on G. Behrmann et al. “A tutorial on UPPAAL”, 2006, str. 36—37)

A timer with a constant time of activeness:

- The timer is activated by the channel (*chan*) *set*.
- At the moment of activation of the timer the clock *x* is set to zero.
- The <u>constant</u> (*const int*) *time* designates length of timer's activity time.
- The variable (*bool*) *a* shows activeness of the timer.
- At the moment of deactivation of the timer the variable *a* becomes false.
- Checking if the counted time has passed: *!a*.
- It is possible to reset the timer during its work.



Modelling patterns

Good practices

- In order to reduce the state space in verifications of formulas it is recommended:
  - to declare variables with a maximally reduced range of values;
  - to reset a local variable to zero, when its value is not needed any more.

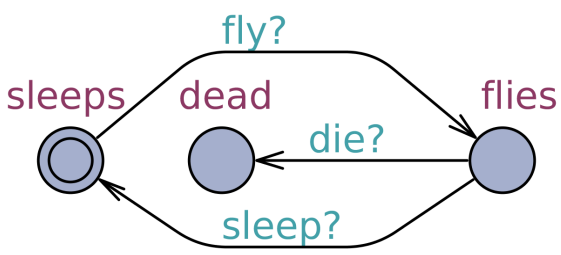
Examples

- Mosquito
- Three escapees
- Petterson's algorithm
- Access to a file
- Trains and a gate

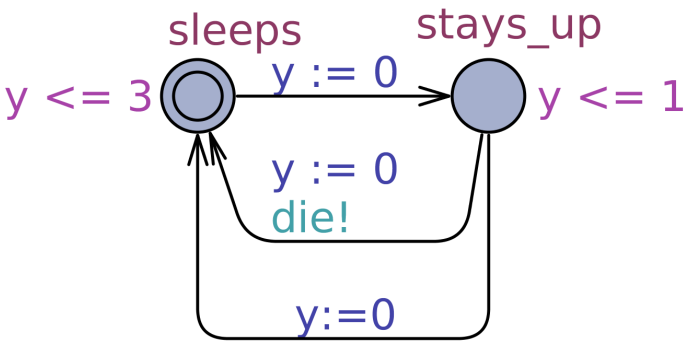
Examples

Mosquito

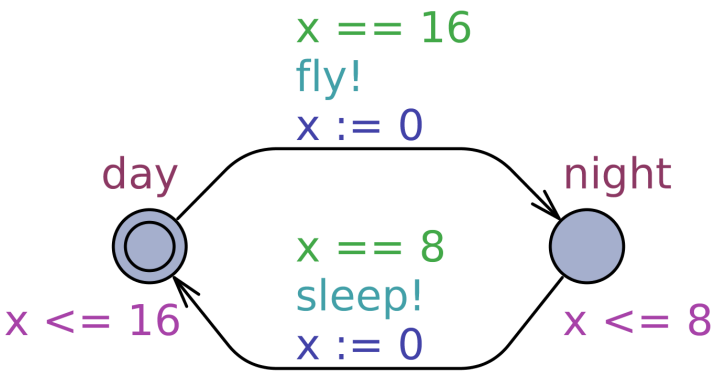
- In summer, when a day lasts 16 hours, and a night lasts 8 hours, a mosquito hunts for blood of a tired man.
- The man sleeps up to 3 hours or stays up up to 1 hour.
- The time makes the mosquito sleep by day, and fly by night.
- When the mosquito flies, and the man stays up, the mosquito may die.



Mosquito



Man

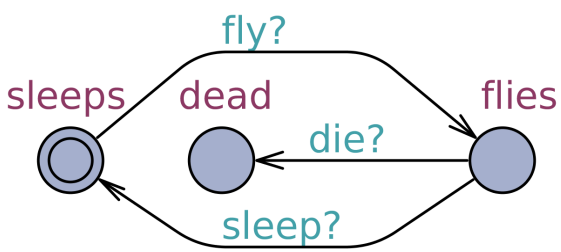


Time

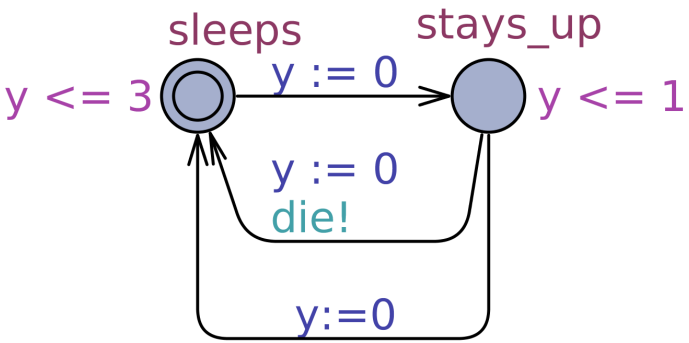
Examples

Mosquito

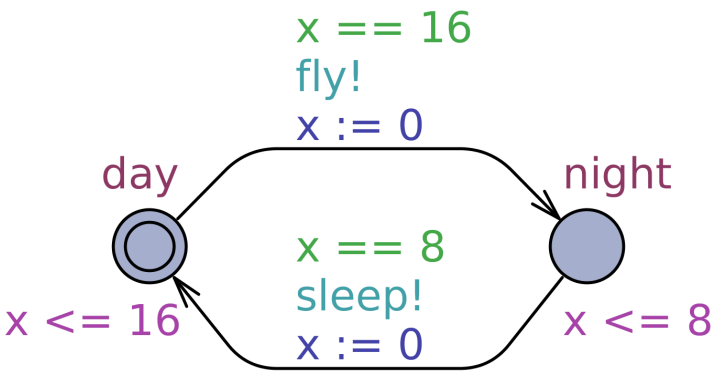
- Can the mosquito be killed?
  E<>Mosquito.dead
- Will the mosquito be certainly killed?
  A<>Mosquito.dead
- Will the flying mosquito be certainly killed?
  Mosquito.flies --> Mosquito.dead
- Does the sleeping mosquito not die?
  A[] Mosquito.sleeps imply not Mosquito.dead



Mosquito



Man

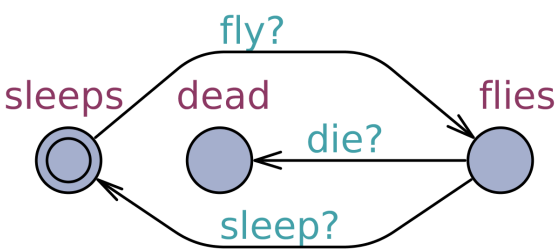


Time

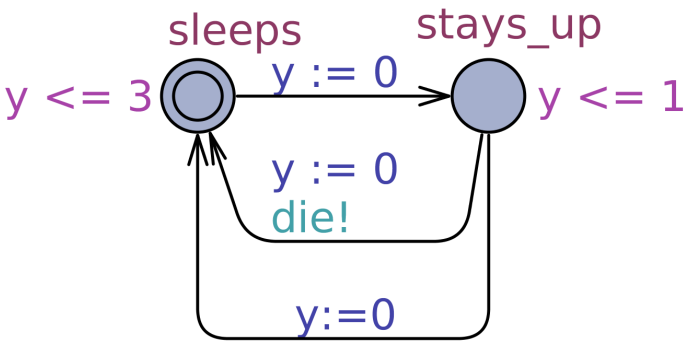
Examples

Mosquito

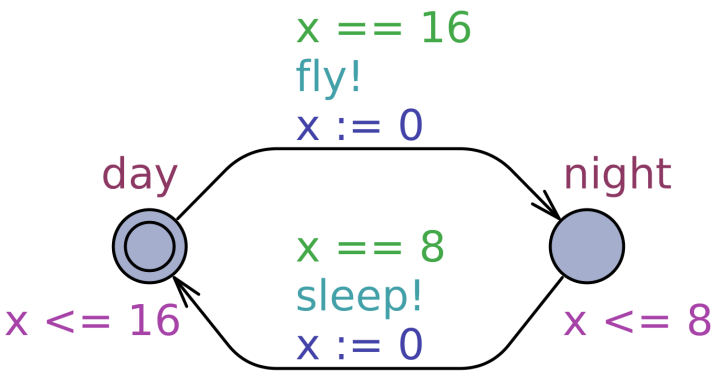
- Can the mosquito be killed?
  E<>Mosquito.dead YES
- Will the mosquito be certainly killed?
  A<>Mosquito.dead
- Will the flying mosquito be certainly killed?
  Mosquito.flies --> Mosquito.dead
- Does the sleeping mosquito not die?
  A[] Mosquito.sleeps imply not Mosquito.dead



Mosquito



Man

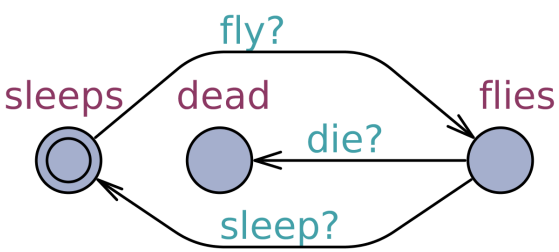


Time

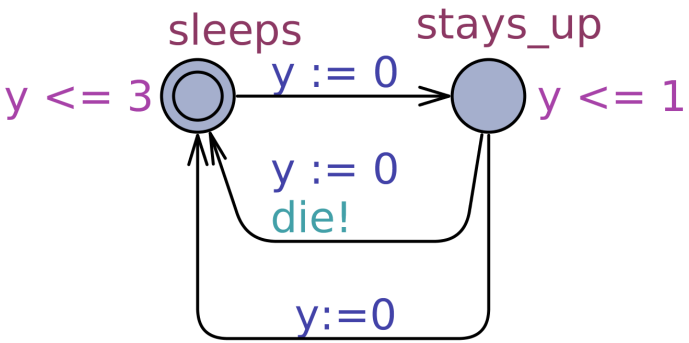
Examples

Mosquito

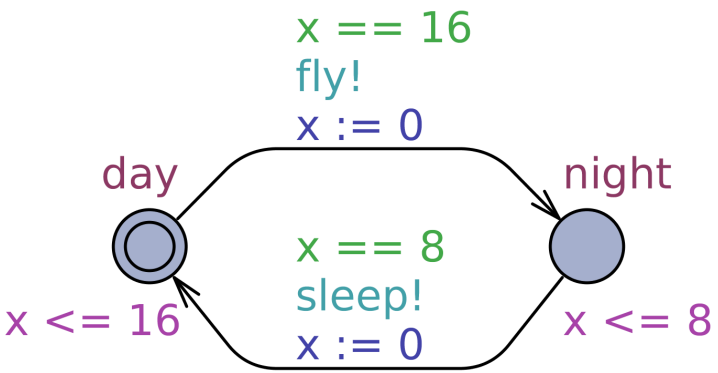
- Can the mosquito be killed?
  E<>Mosquito.dead YES
- Will the mosquito be certainly killed?
  A<>Mosquito.dead NO
- Will the flying mosquito be certainly killed?
  Mosquito.flies --> Mosquito.dead
- Does the sleeping mosquito not die?
  A[] Mosquito.sleeps imply not Mosquito.dead



Mosquito



Man

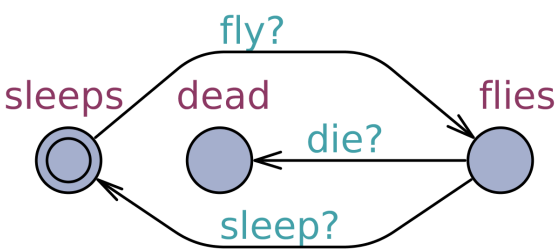


Time

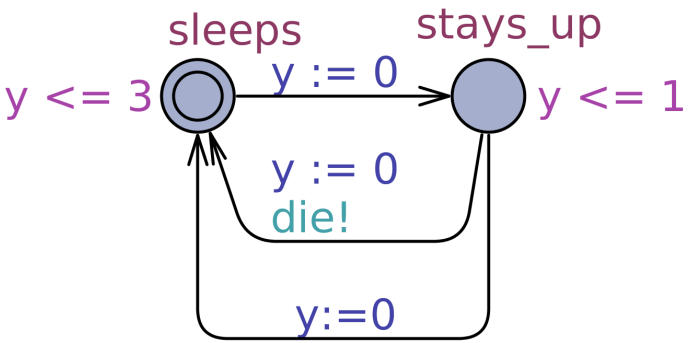
Examples

Mosquito

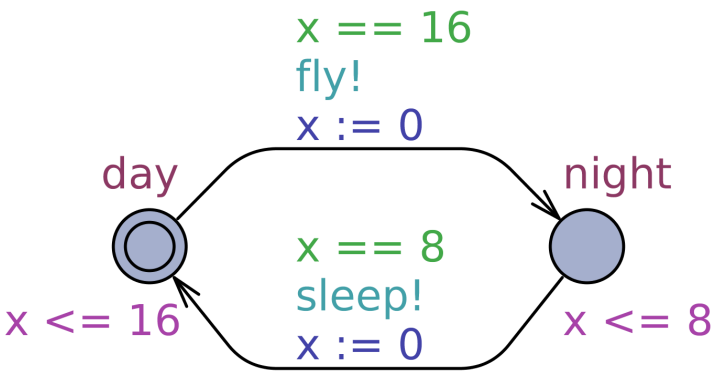
- Can the mosquito be killed?
  E<>Mosquito.dead YES
- Will the mosquito be certainly killed?
  A<>Mosquito.dead NO
- Will the flying mosquito be certainly killed?
  Mosquito.flies --> Mosquito.dead NO
- Does the sleeping mosquito not die?
  A[] Mosquito.sleeps imply not Mosquito.dead



Mosquito



Man

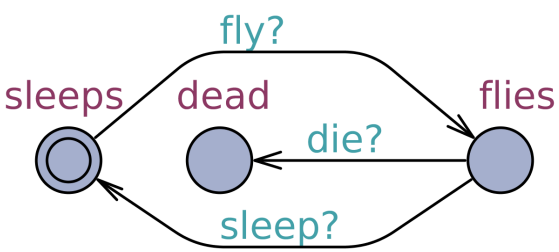


Time

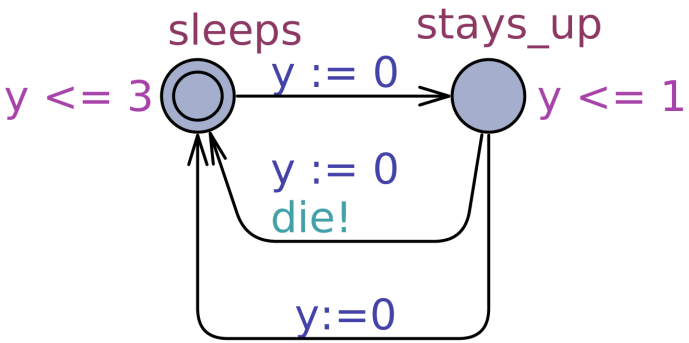
Examples

Mosquito

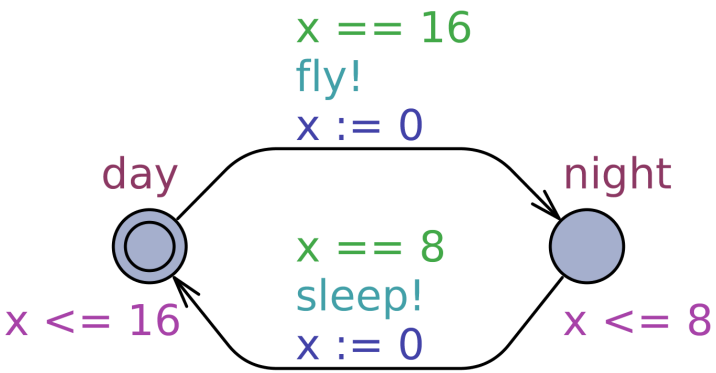
- Can the mosquito be killed?
  E<>Mosquito.dead YES
- Will the mosquito be certainly killed?
  A<>Mosquito.dead NO
- Will the flying mosquito be certainly killed?
  Mosquito.flies --> Mosquito.dead NO
- Does the sleeping mosquito not die?
  A[] Mosquito.sleeps imply not Mosquito.dead YES



Mosquito



Man



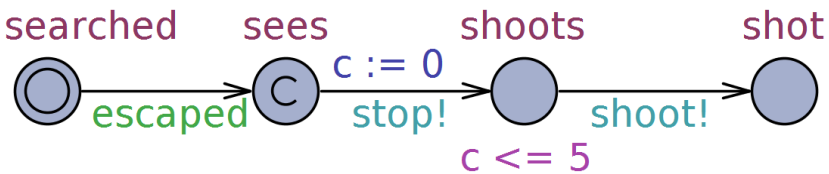
Time

Examples

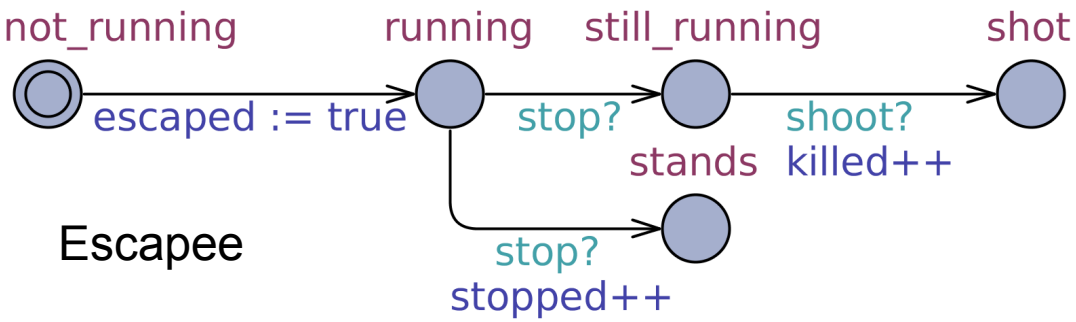
Three escapees

Three escapees escaped from a prison.

- They are searched for by one policeman.
- When he sees the first of them, he shouts *stop!* (*broadcast chan*).
- Every running escapee may stop.
- Exactly one of the still running escapees will be killed by a shot of the policeman.
- The policeman has at most 5 seconds to shoot.



Policeman



Escapee

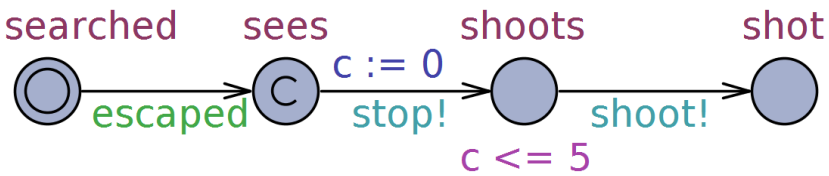
Examples

Three escapees

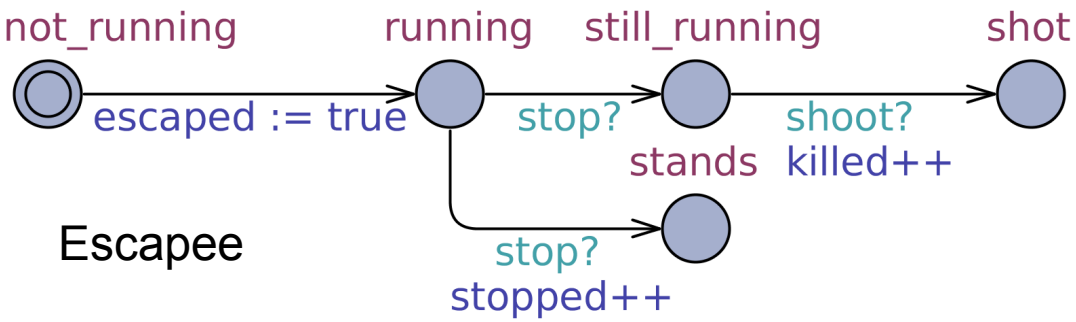
Three escapees escaped from a prison.

- Will all escapees stop?
  A<> forall (i:int[1,3]) Escapee(i).stops
  A<> stopped == 3

- Is it possible to stop all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).stops
  E<> stopped == 3



Policeman



Escapee

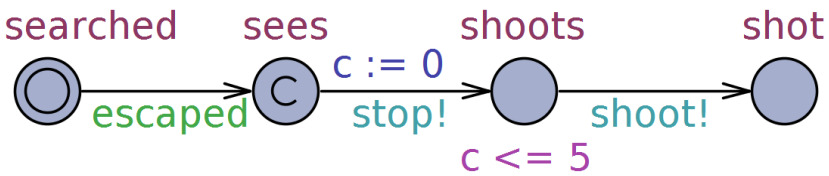
Examples

Three escapees

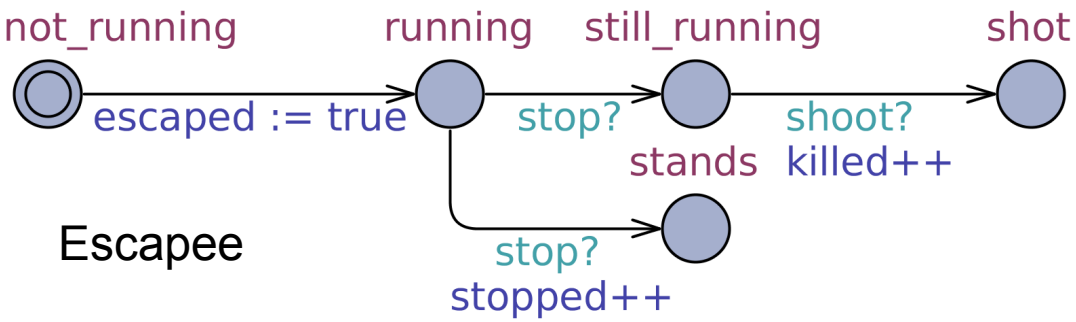
Three escapees escaped from a prison.

- Will all escapees stop?
  A<> forall (i:int[1,3]) Escapee(i).stops
  A<> stopped == 3
  NO

- Is it possible to stop all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).stops
  E<> stopped == 3



Policeman



Escapee

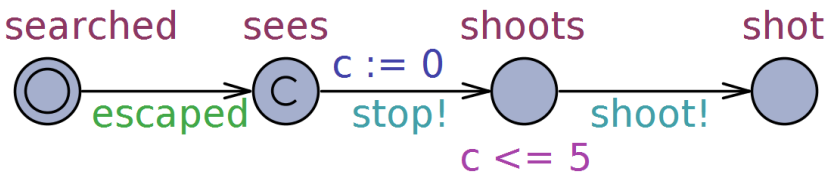
Examples

Three escapees

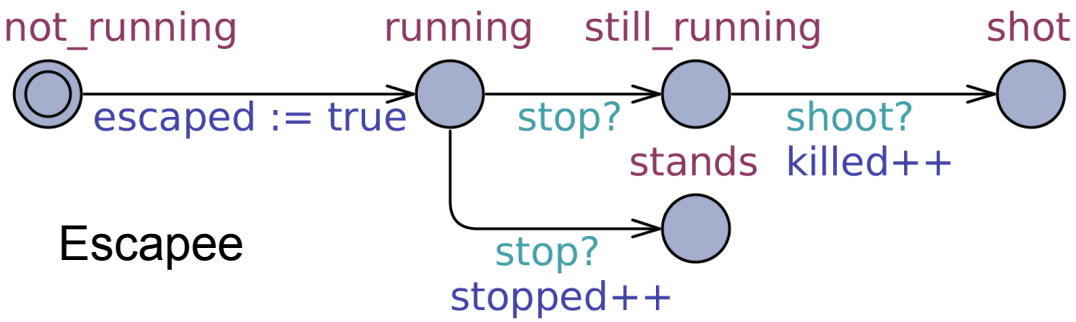
Three escapees escaped from a prison.

- Will all escapees stop?
  A<> forall (i:int[1,3]) Escapee(i).stops
  A<> stopped == 3
  NO

- Is it possible to stop all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).stops
  E<> stopped == 3
  YES



Policeman



Escapee

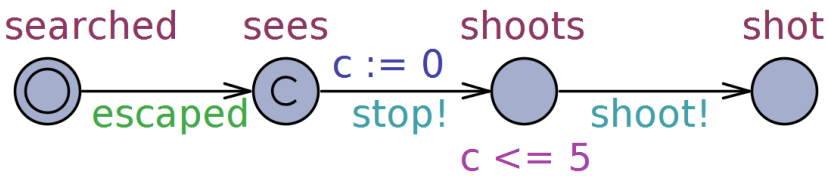
Examples

Three escapees

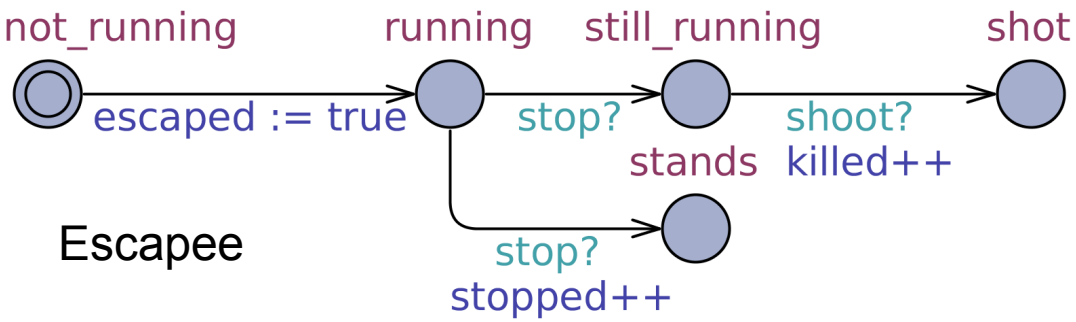
Three escapees escaped from a prison.

- Is it possible to shoot more than one escapee?
  A[] forall (i:int[1,3]) Escapee(i).shot imply killed == 1 counterexample!
  E<> killed > 1

- Is it possible to shoot all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).shot
  E<> killed == 3



Policeman



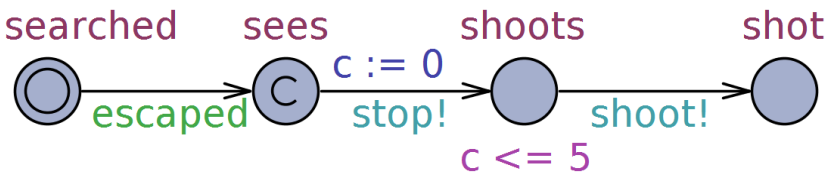
Escapee

Examples

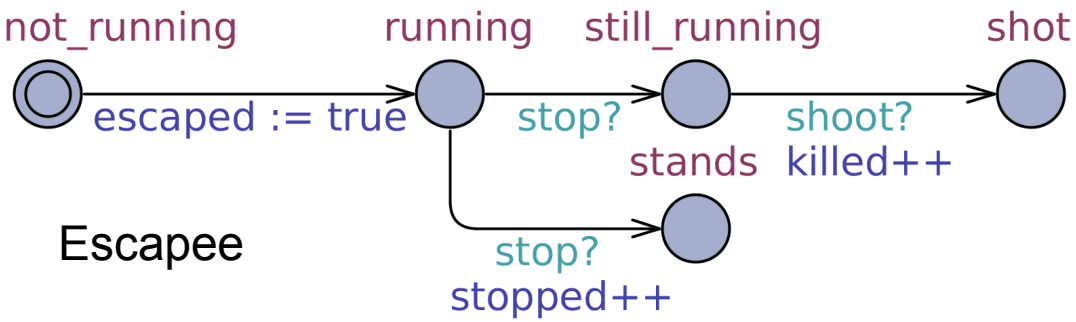
Three escapees

Three escapees escaped from a prison.

- Is it possible to shoot more than one escapee?
  A[] forall (i:int[1,3]) Escapee(i).shot imply killed == 1 counterexample!
  E<> killed > 1
  NO
- Is it possible to shoot all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).shot
  E<> killed == 3



Policeman



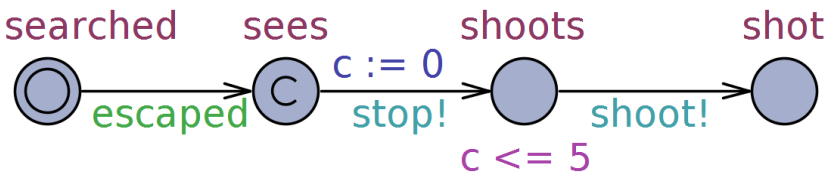
Escapee

Examples

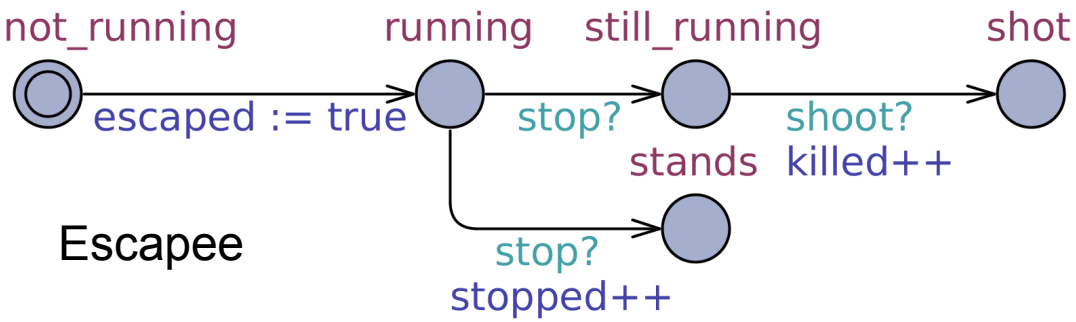
Three escapees

Three escapees escaped from a prison.

- Is it possible to shoot more than one escapee?
  A[] forall (i:int[1,3]) Escapee(i).shot imply killed == 1 counterexample!
  E<> killed > 1
  NO
- Is it possible to shoot all the escapees?
  E<> forall (i:int[1,3]) Escapee(i).shot
  E<> killed == 3
  NO



Policeman



Escapee

Examples

Petterson's algorithm

(based on A. David et al. “UPPAAL 4.0: Small tutorial”, 2009)

The Petterson's algorithm for mutual exclusion of 2 processes:

- **Process P1:**

```
req1=1;                          // state “idle”
turn=2;                          // state “want”
while(turn!=1 && req2!=0);       // state “wait”
req1=0;                          // state “CS”
```

- **Process P2:**

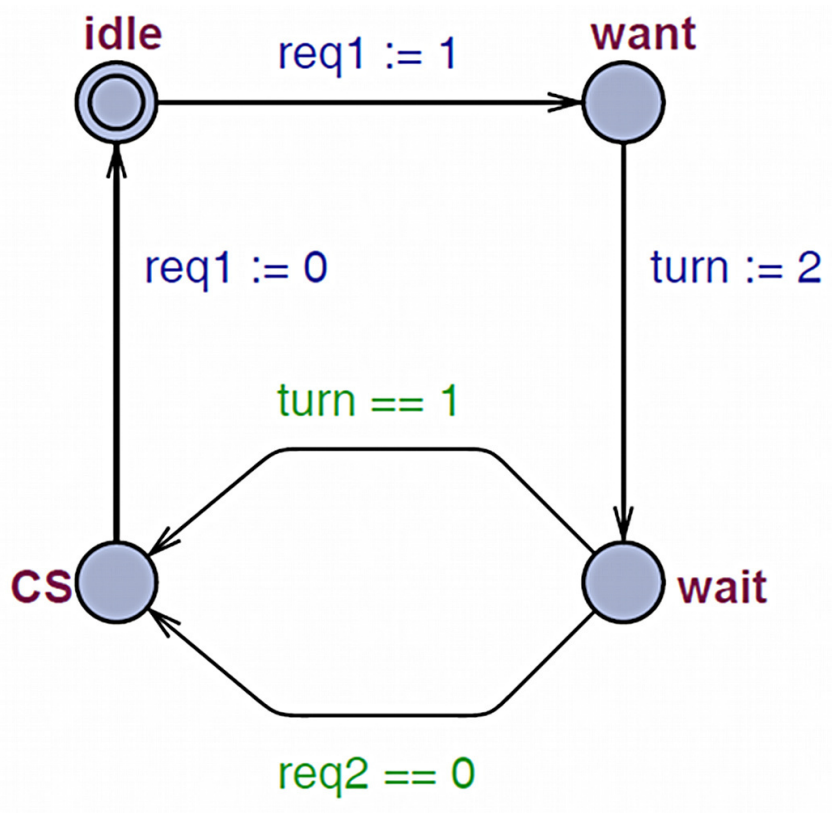
```
req2=1;                          // state “idle”
turn=1;                          // state “want”
while(turn!=2 && req1!=0);       // state “wait”
req2=0;                          // state “CS”
```

req1, req2, turn — control variables
CS — critical section

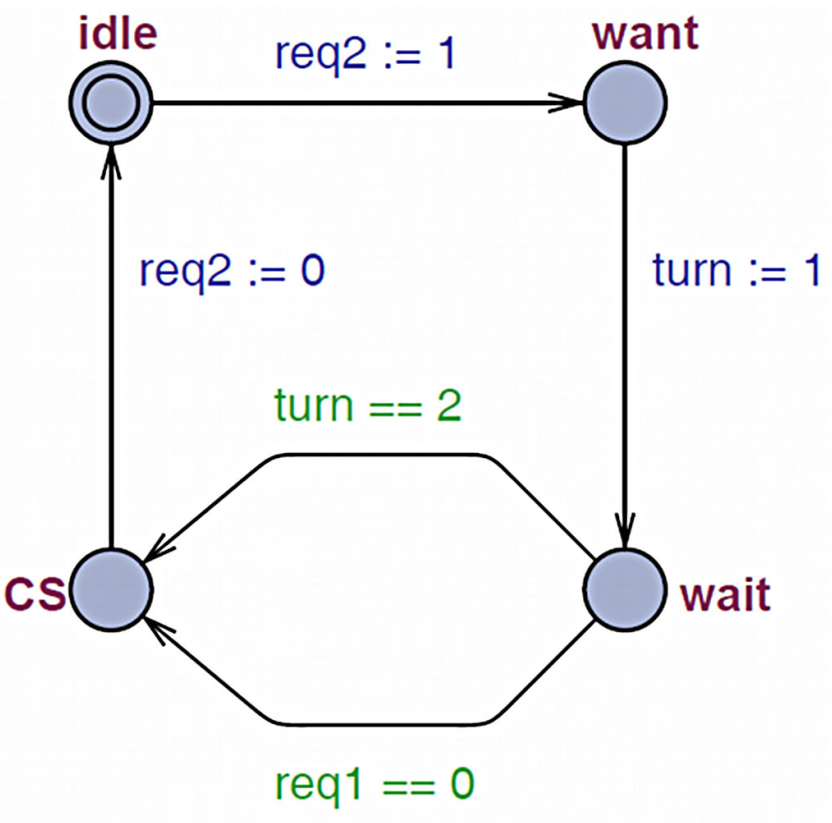
Examples

Petterson's algorithm

(based on A. David et al. “UPPAAL 4.0: Small tutorial”, 2009)



P1



P2

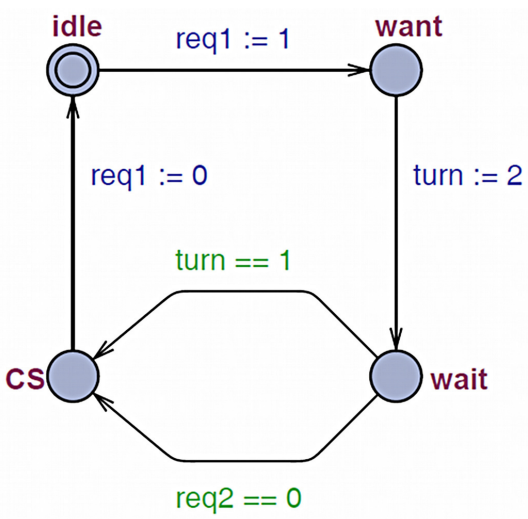
Examples

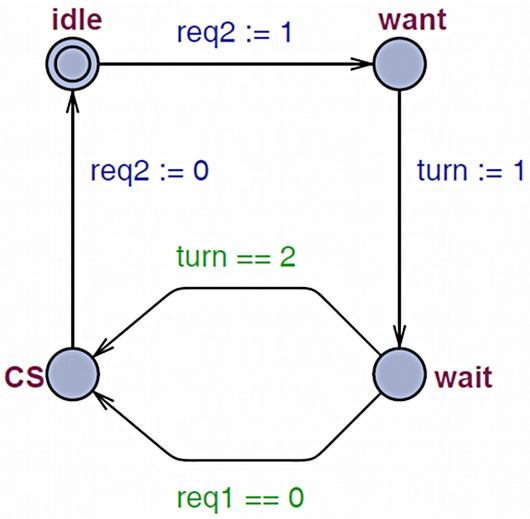
Petterson's algorithm

(based on A. David et al. “UPPAAL 4.0: Small tutorial”, 2009)

Properties related to the critical section (CS):

- *safety* — mutual exclusion of both the processes:
  A[] not (P1.CS and P2.CS)
- *reachability* — access to the critical section:
  E<>(P1.CS)
  E<>(P2.CS)





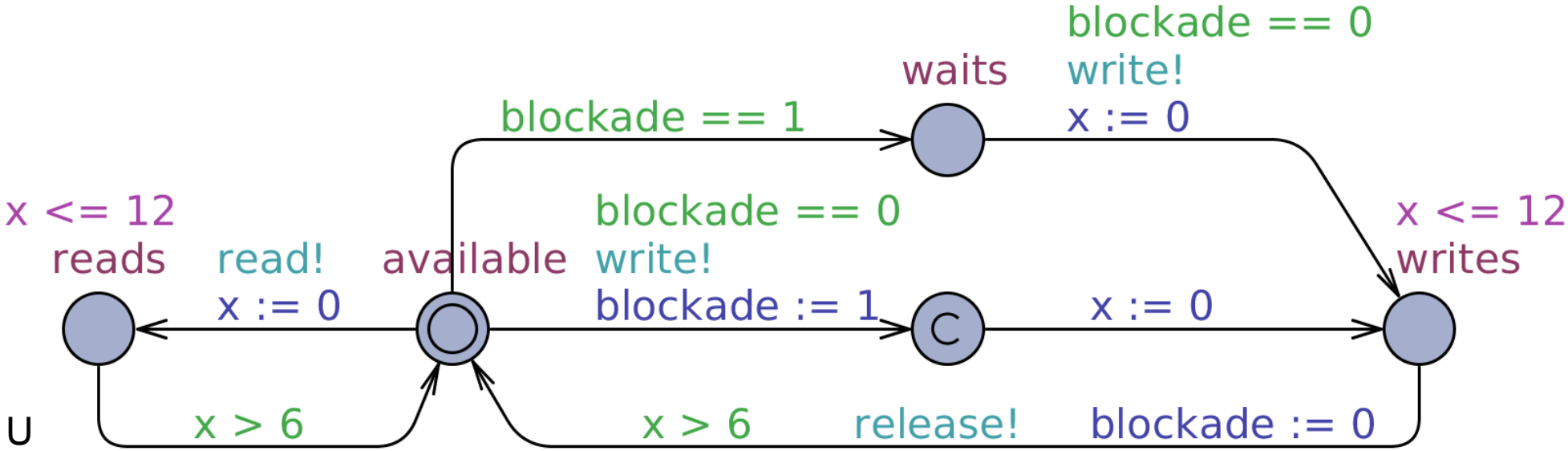
Examples

Access to a file

Users *U(0)* and *U(1)* may want to access a file:

- The file is always available to be read.
- The file is available to be written to for one user only at once. The other must waits for the access then.

File

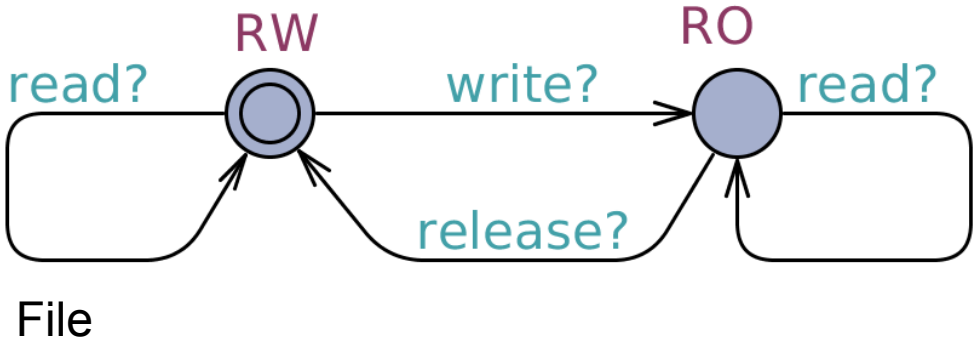


U

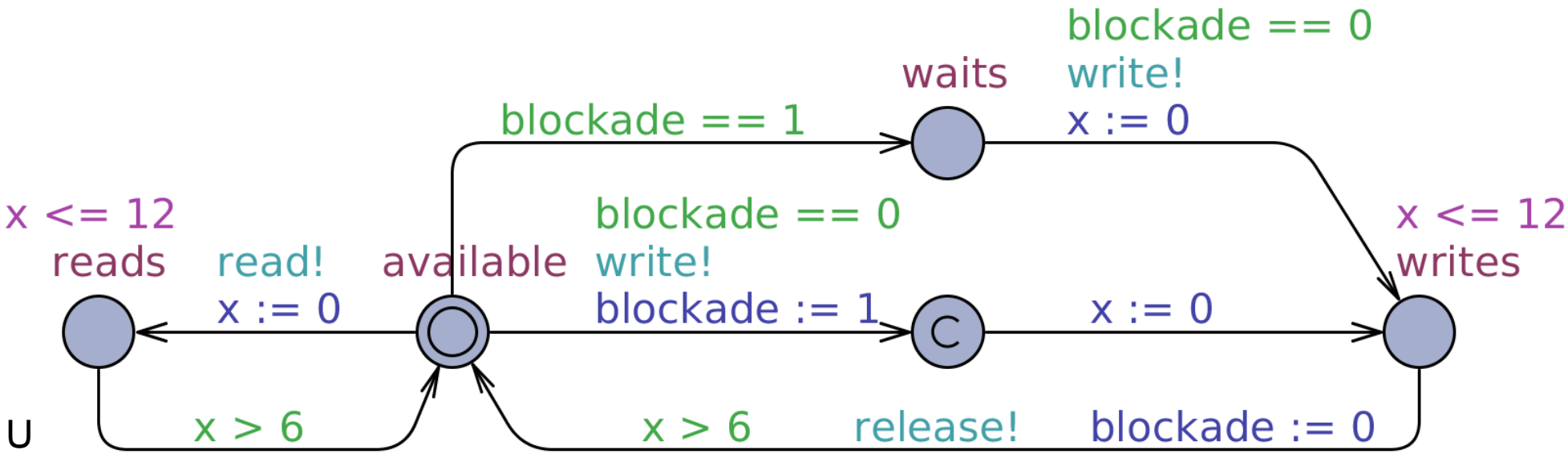
Examples

Access to a file

- Can U(0) write, while U(1) is waiting?
  E<> U(0).writes and U(1).waits



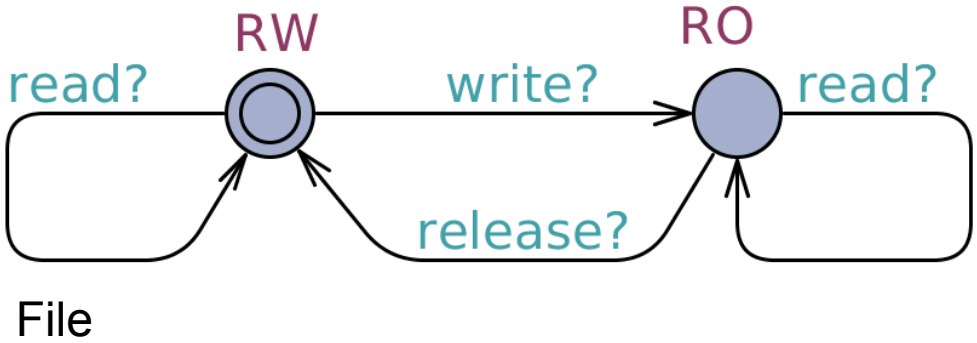
- Can the file be written to by more than one user at once?
  A[] forall (i : int[0,1]) forall (j : int[0,1]) U(i).writes && U(j).writes imply i == j

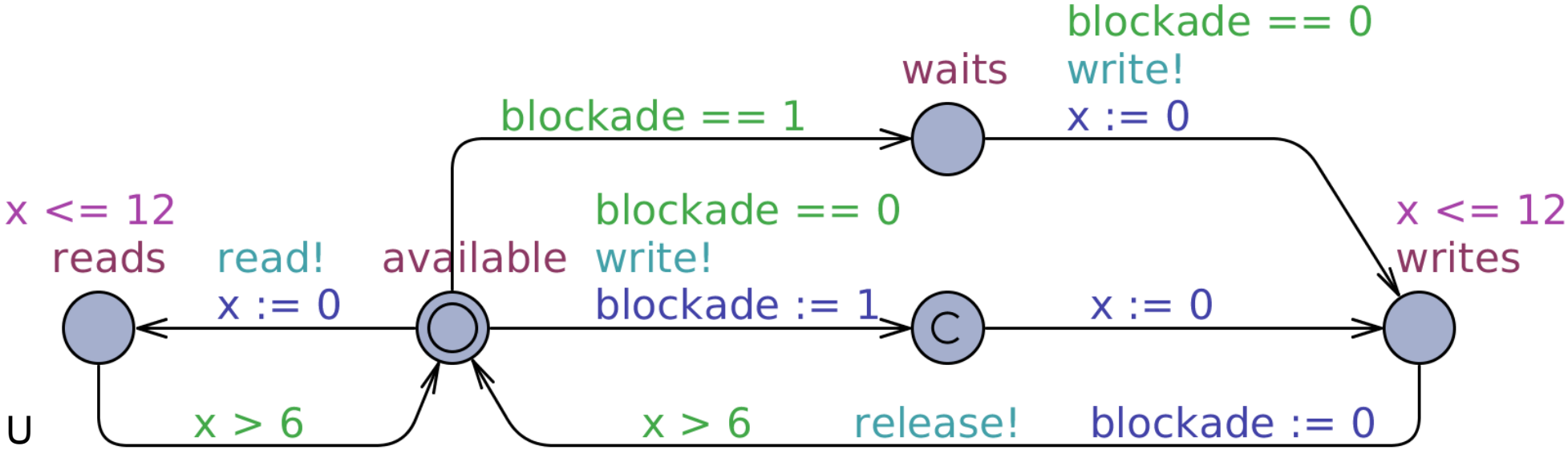


Examples

Access to a file

- Can U(0) write, while U(1) is waiting?
  E<> U(0).writes and U(1).waits
  YES
- Can the file be written to by more than one user at once?
  A[] forall (i : int[0,1]) forall (j : int[0,1]) U(i).writes && U(j).writes imply i == j

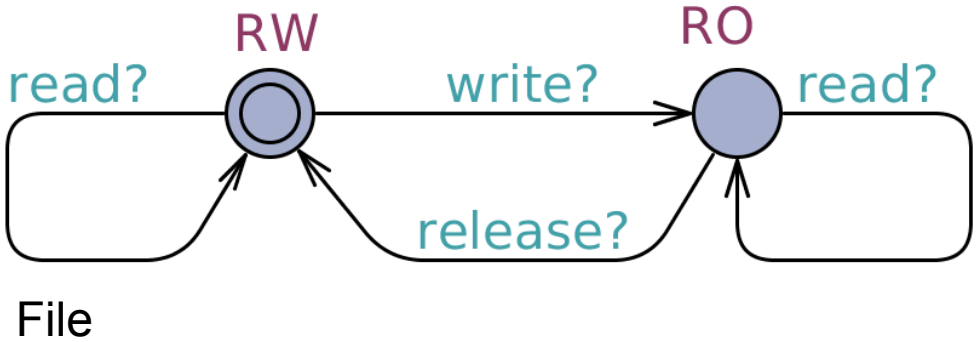




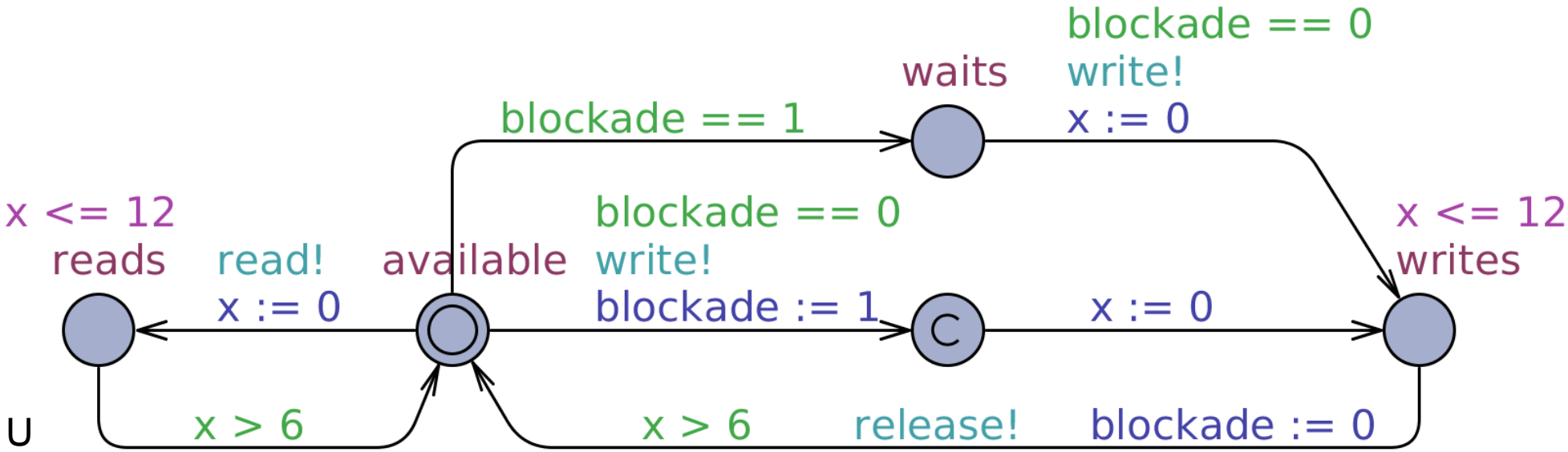
Examples

Access to a file

- Can U(0) write, while U(1) is waiting?
  E<> U(0).writes and U(1).waits
  YES



- Can the file be written to by more than one user at once?
  A[] forall (i : int[0,1]) forall (j : int[0,1]) U(i).writes && U(j).writes imply i == j
  NO

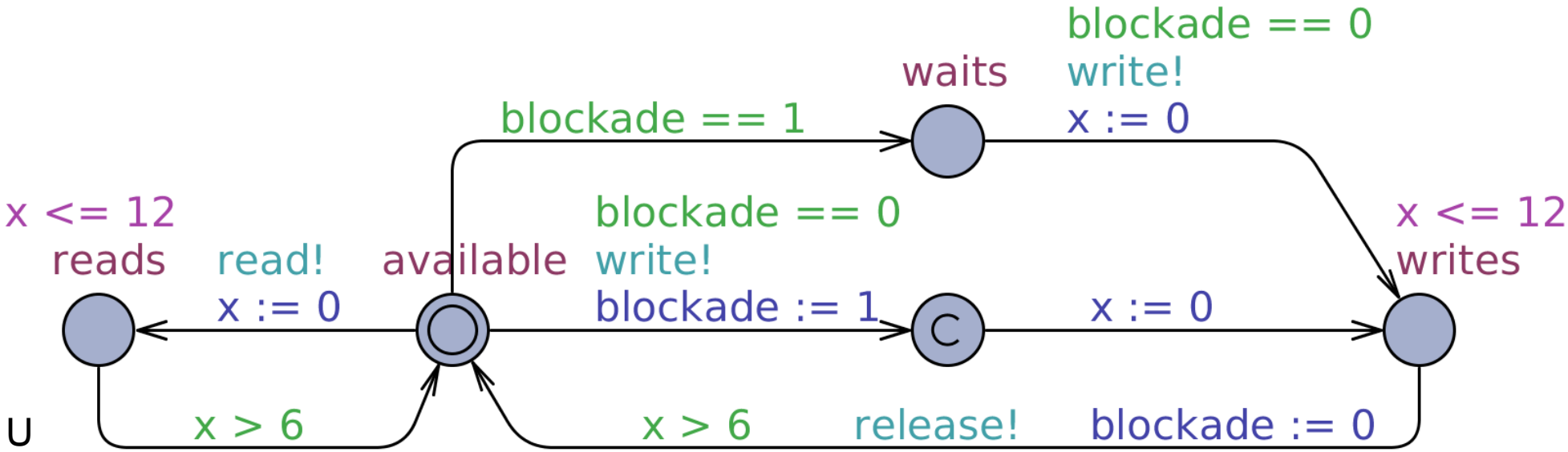


Examples

Access to a file

- Will the waiting user (e.g. U(0)) certainly get access to write?
  U(0).waits --> U(0).writes

File



U

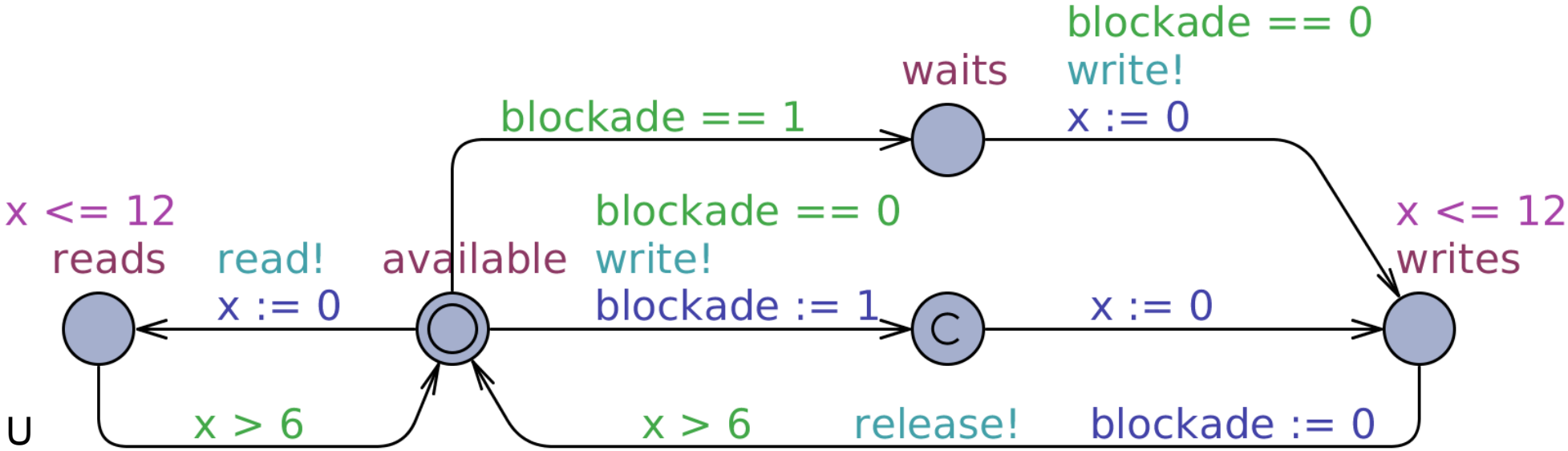
Examples

Access to a file

- Will the waiting user (e.g. U(0)) certainly get access to write?
  U(0).waits --> U(0).writes
  NO

File

- **Waiting for the access may be infinite.**
  How to fix it?



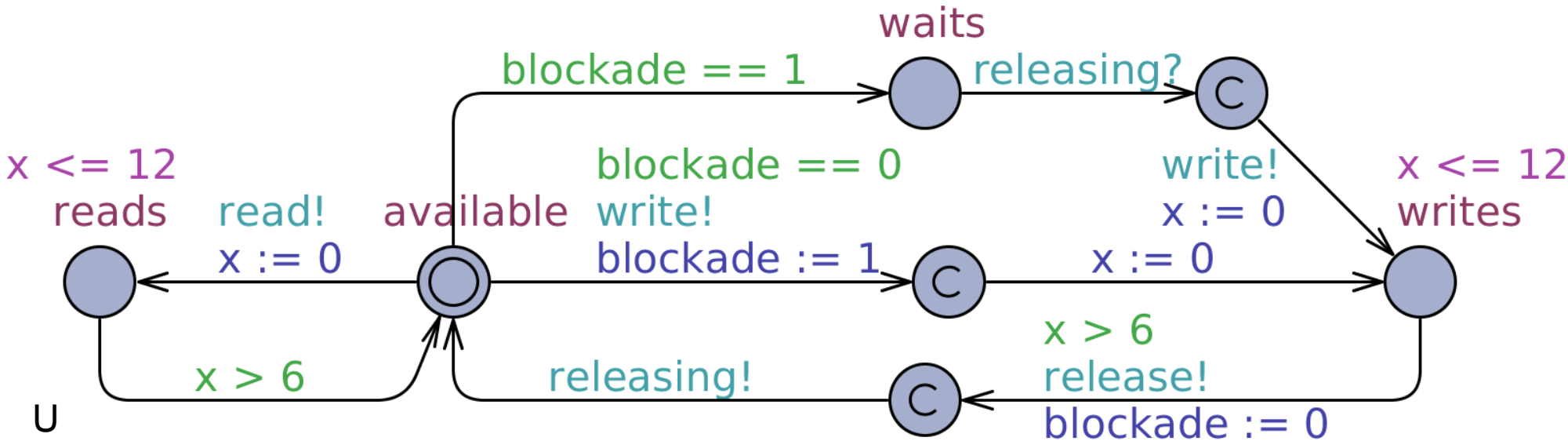
U

Examples

Access to a file

- Will the waiting user (e.g. U(0)) certainly get access to write?
  U(0).waits --> U(0).writes
  YES

File



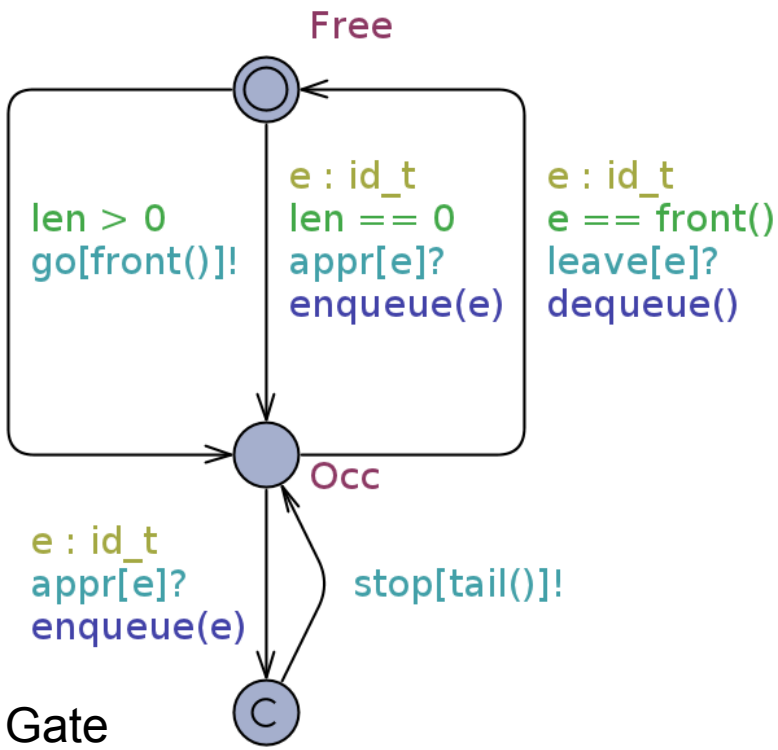
U

Examples

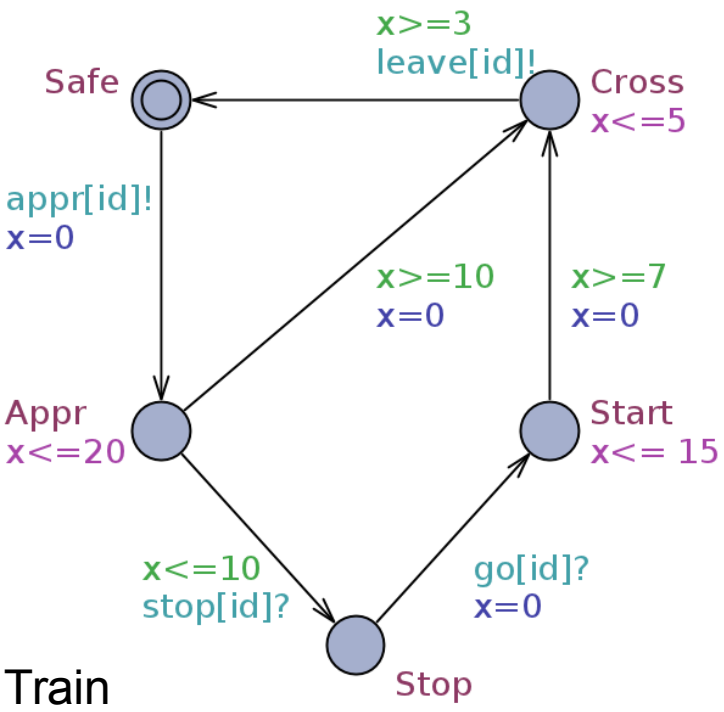
Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- N trains are approaching a passage through a bridge at once.
- A gate lets at most one train at once to the bridge, telling other trains to wait for their turn.



Gate



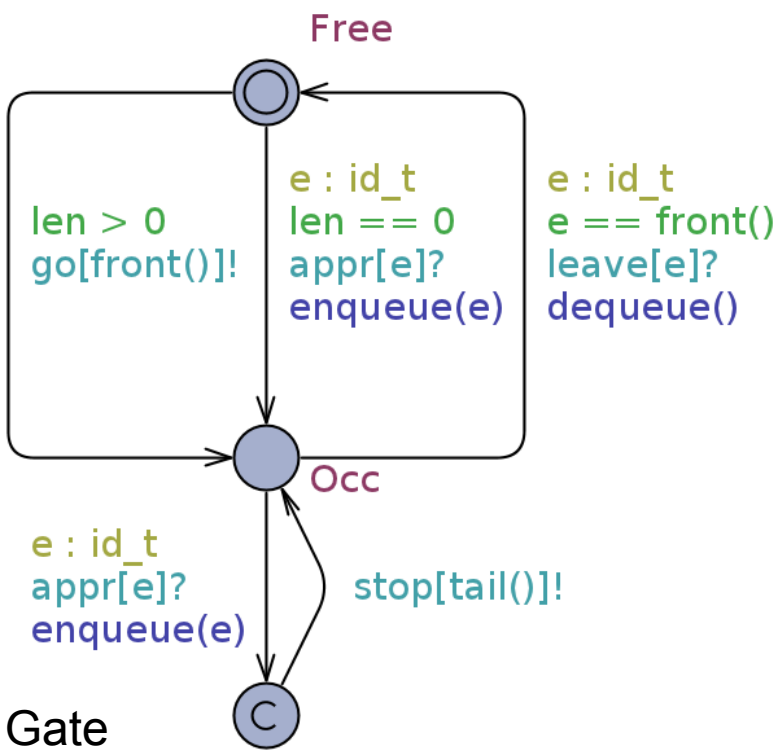
Train

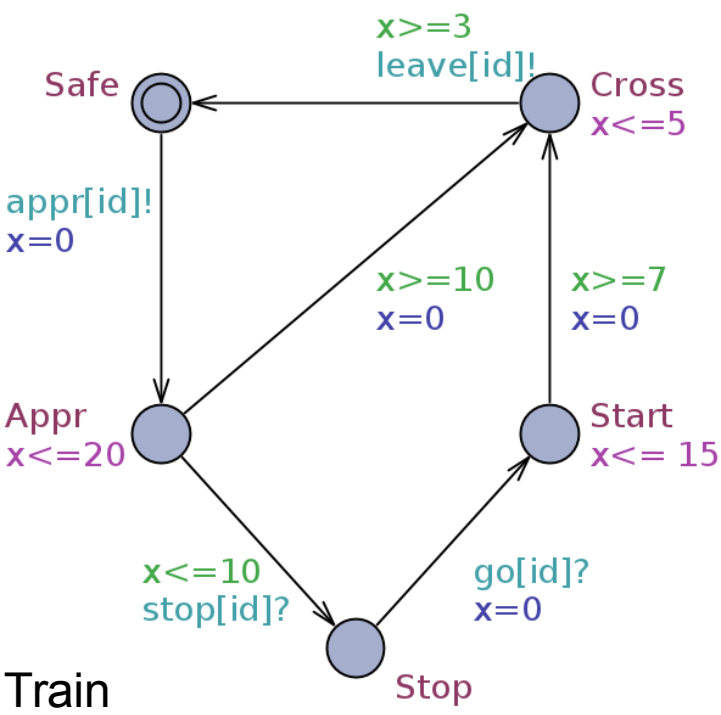
Examples

Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- Changing states of a train takes some time
  (x – clock variable)





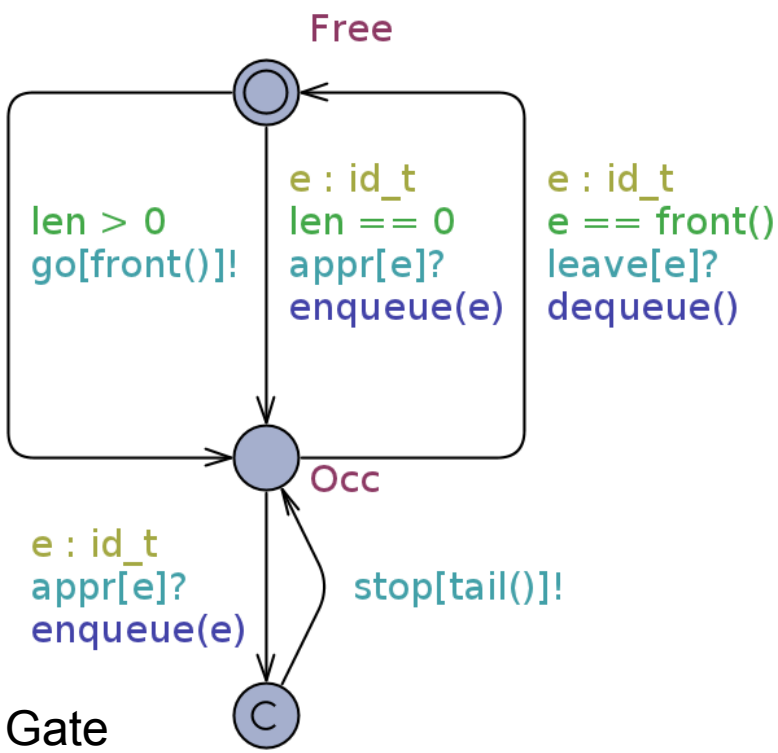
Examples

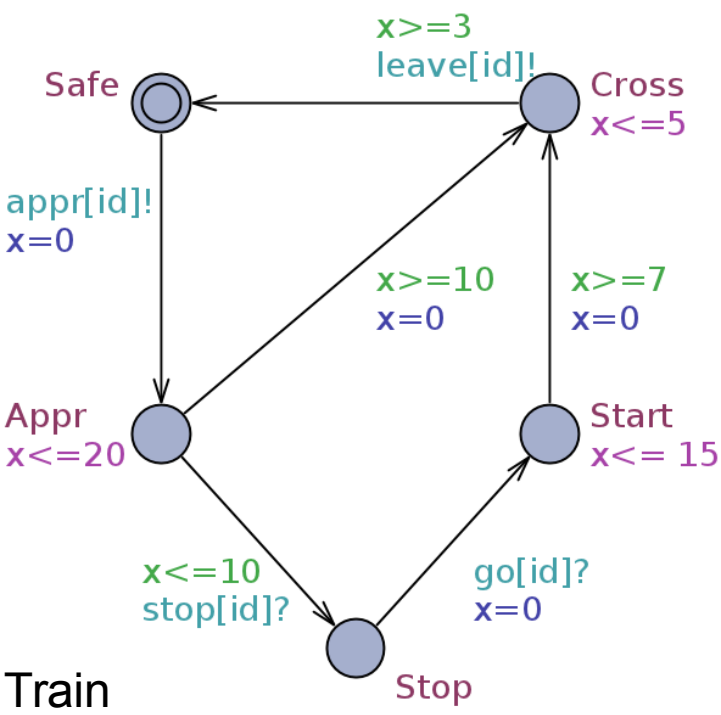
Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- The gate may receive (and store in a queue) messages from approaching trains:

E<> Gate.Occ





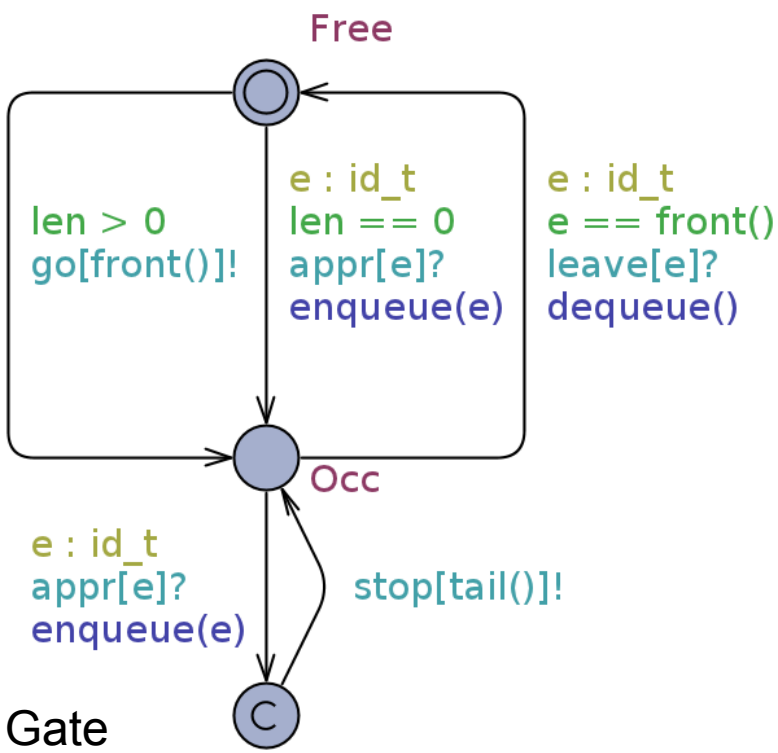
Examples

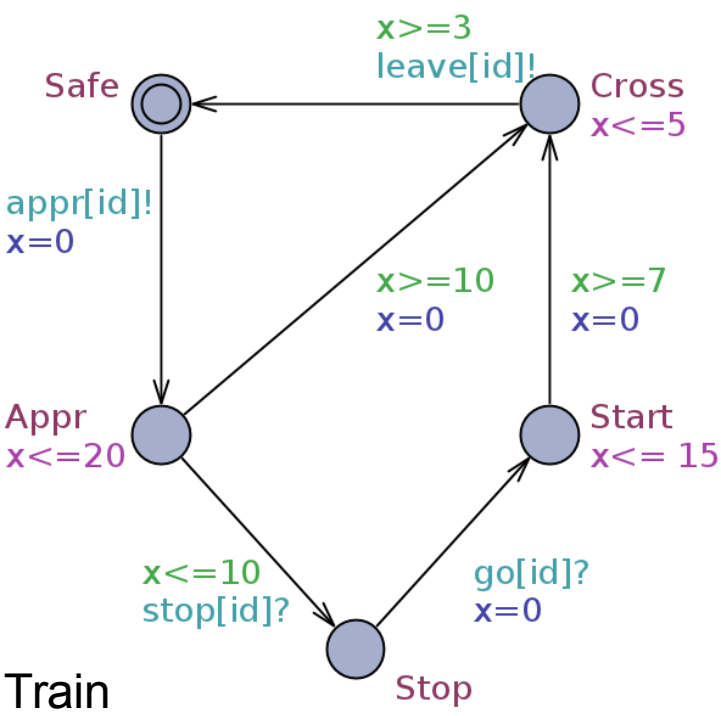
Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- Train 0 may cross the bridge, while other trains are waiting to cross it:

  E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)





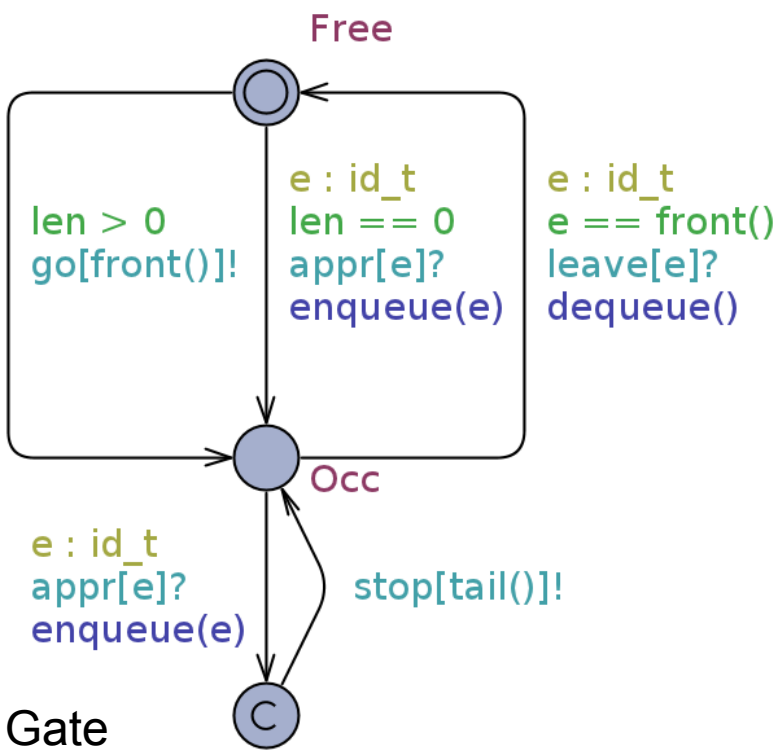
Examples

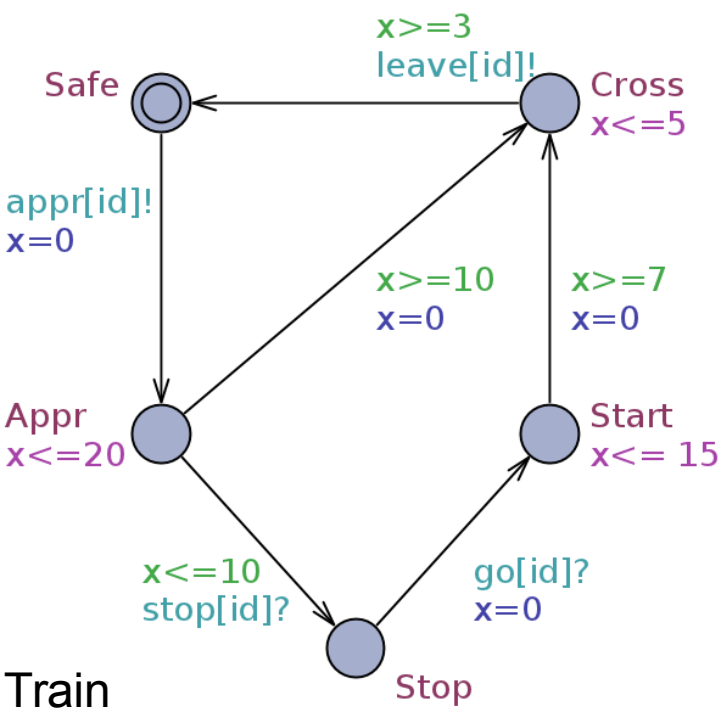
Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- Never more then one train crosses the bridge at once:

  A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j





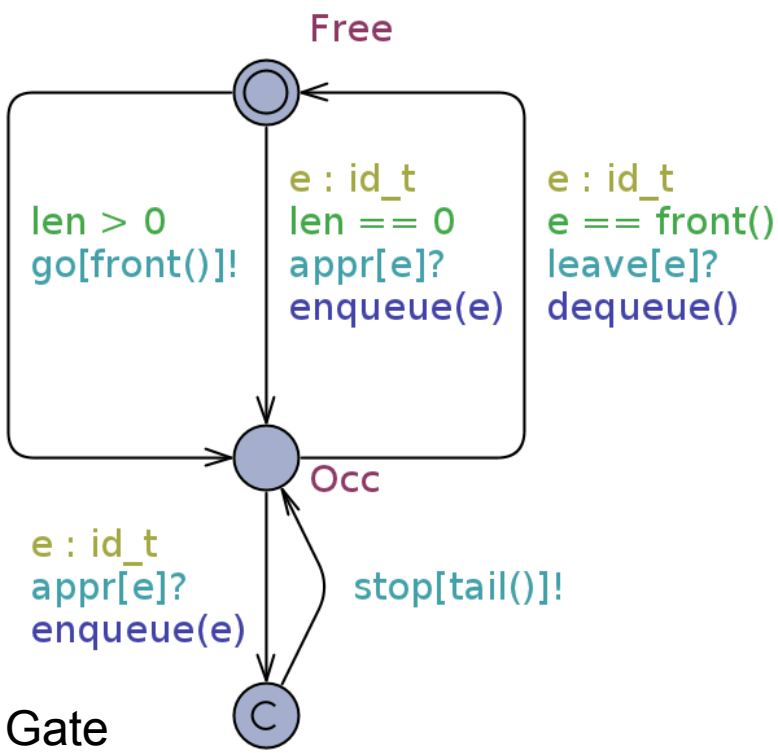
Examples

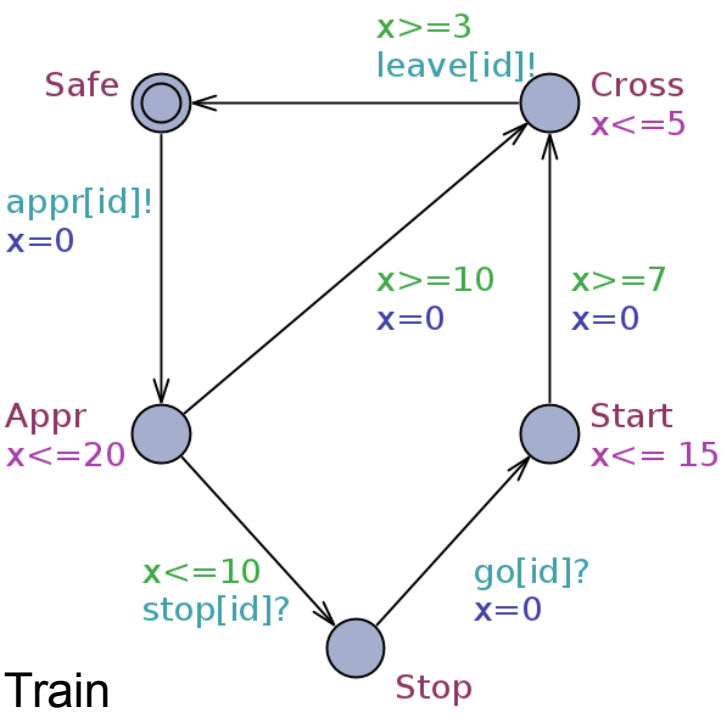
Trains and a gate

(based on W. Yi et al. “Automatic Verification of Real-Time Communicating Systems by Constraint Solving”,1994.)
This model is in the demo catalogue of the UPPAAL program.

- Whenever a train (e.g. 0) approaches the bridge, eventually it will cross it:

  Train(0).Appr --> Train(0).Cross





The end

Literature:

- G. Behrmann et al. “A tutorial on UPPAAL”, 2006, at: www.uppaal.com
- A. David et al. “UPPAAL 4.0: Small tutorial”, 2009, at: www.uppaal.com
- “UPPAAL Language Reference”, http://www.uppaal.com/index.php?sida=217&rubrik=101