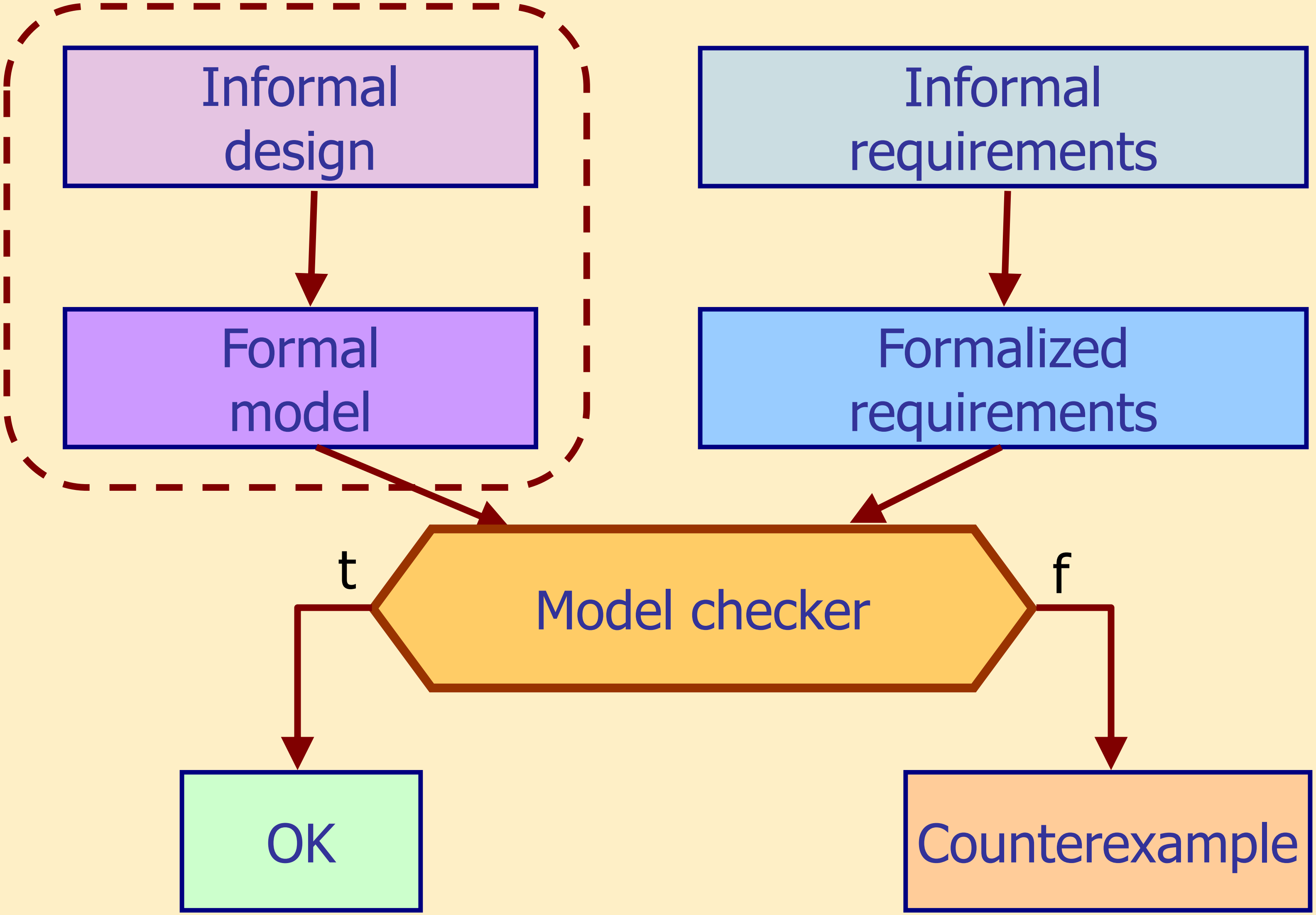
Basic Formalisms

dr. István Majzik

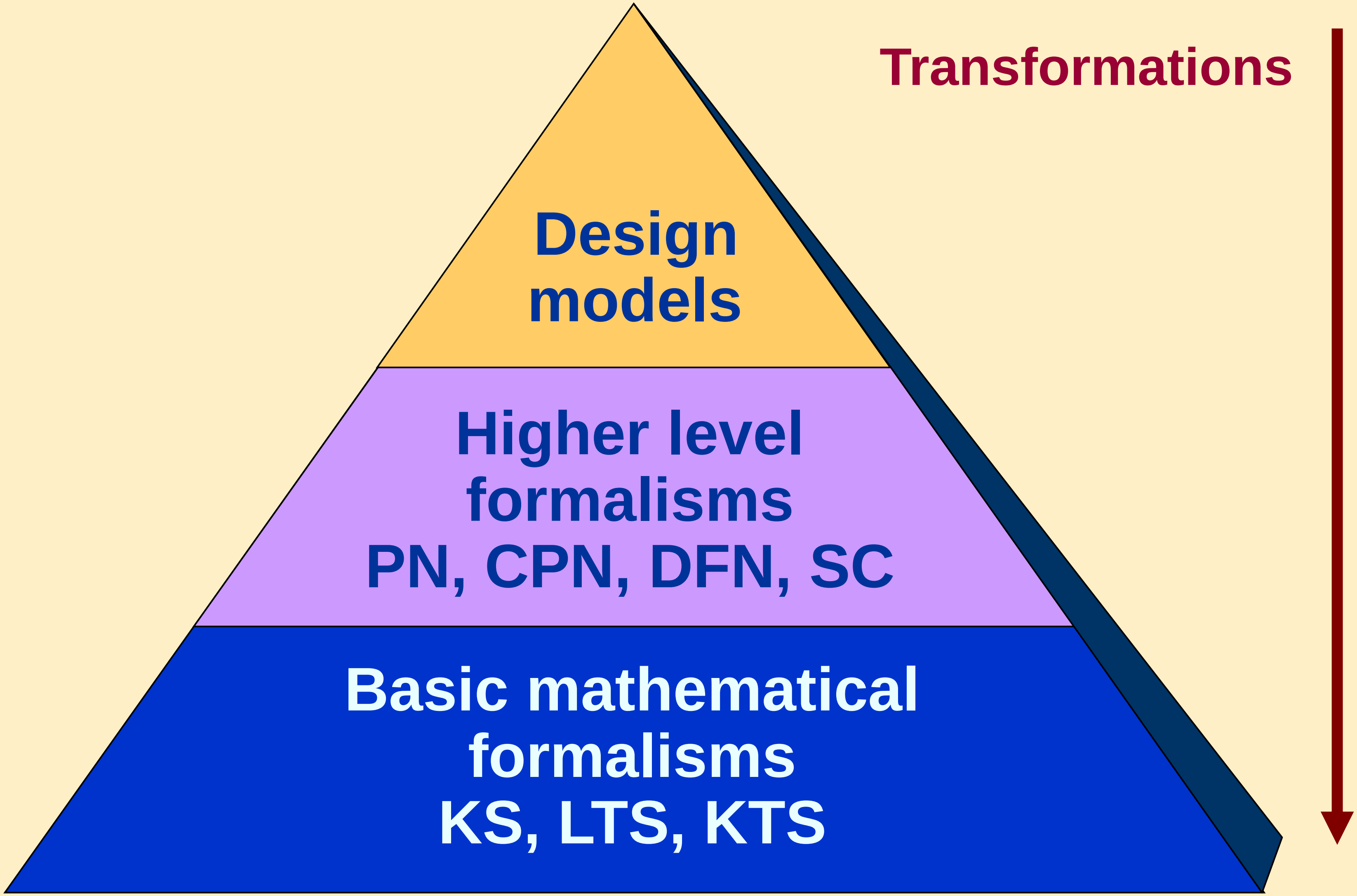
BME Department of Measurement and Information Systems

Our goal



Informal design
Informal requirements
Formal model
Formalized requirements
t
Model checker
f
OK
Counterexample

Formalisms for formal verification



Basic fomalisms (overview)

- Kripke Structures (KS)
  - States, transitions, labels
  - Local properties of states as labels
- Labeled Transition Systems (LTS)
  - States, transitions, actions
  - Local properties of transitions as actions
- Kripke Transition Systems (KTS)
  - States, transitions, labels, actions
  - Local properties of states and transitions as labels and actions
- Finite State Automata with Time
  - Extensions: variables, clocks, synchronization

Kripke Structure

- Expresses properties of states:
  labeling by atomic propositions
- Possibly more than one labels per state
- Application: description of behavior or algorithm

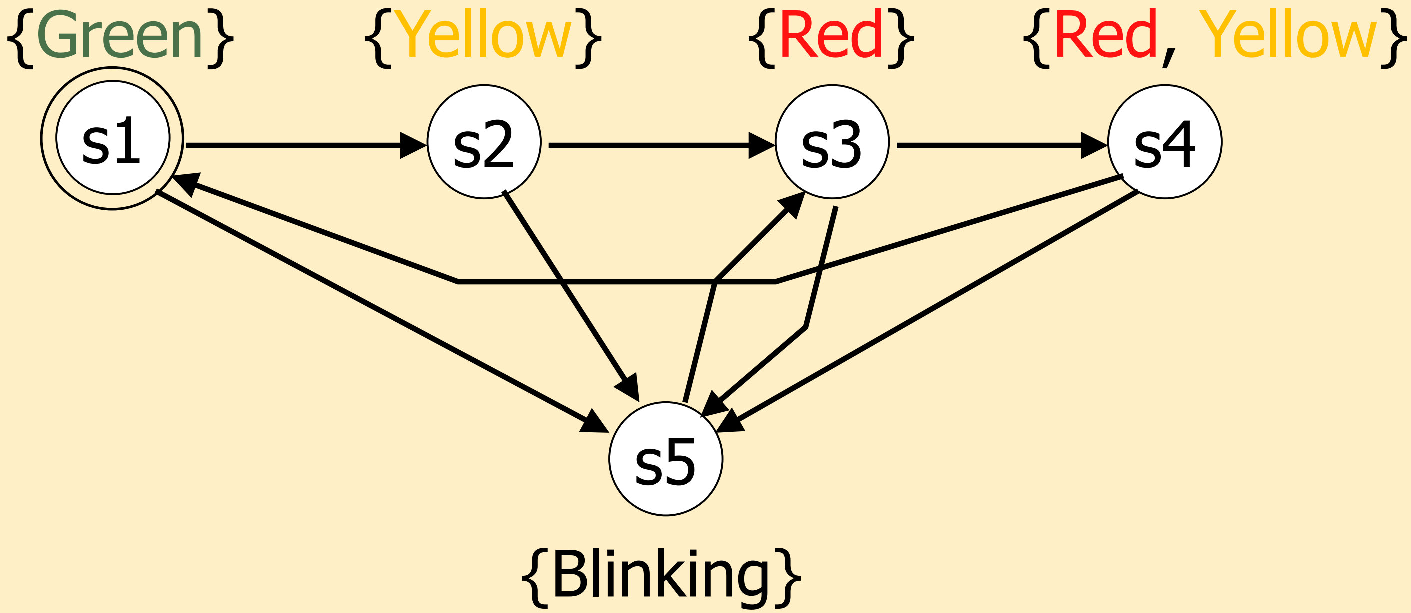
A Kripke structure KS over a set of atomic propositions $AP = \{P, Q, R, \dots\}$ is a tuple (S, I, R, L) where

- $S = \{s_1, s_2, \dots, s_n\}$ is a finite set of states,
- $I \subseteq S$ is the set of initial states,
- $R \subseteq S \times S$ is the set of transitions and
- $L : S \rightarrow 2^{AP}$ is the labeling of states by atomic propositions

Example for KS

Traffic light

- $AP = \{Green, Yellow, Red, Blinking\}$
- $S = \{s_1, s_2, s_3, s_4, s_5\}$



Labeled Transition System

- Expresses properties of transitions:
  labeling by actions
- Exactly one action per transition
- Application: modeling of communication and protocols

A labeled transition system LTS over a set of actions $Act = \{a, b, c, \dots\}$ is a triple $(S, I, \rightarrow)$ where

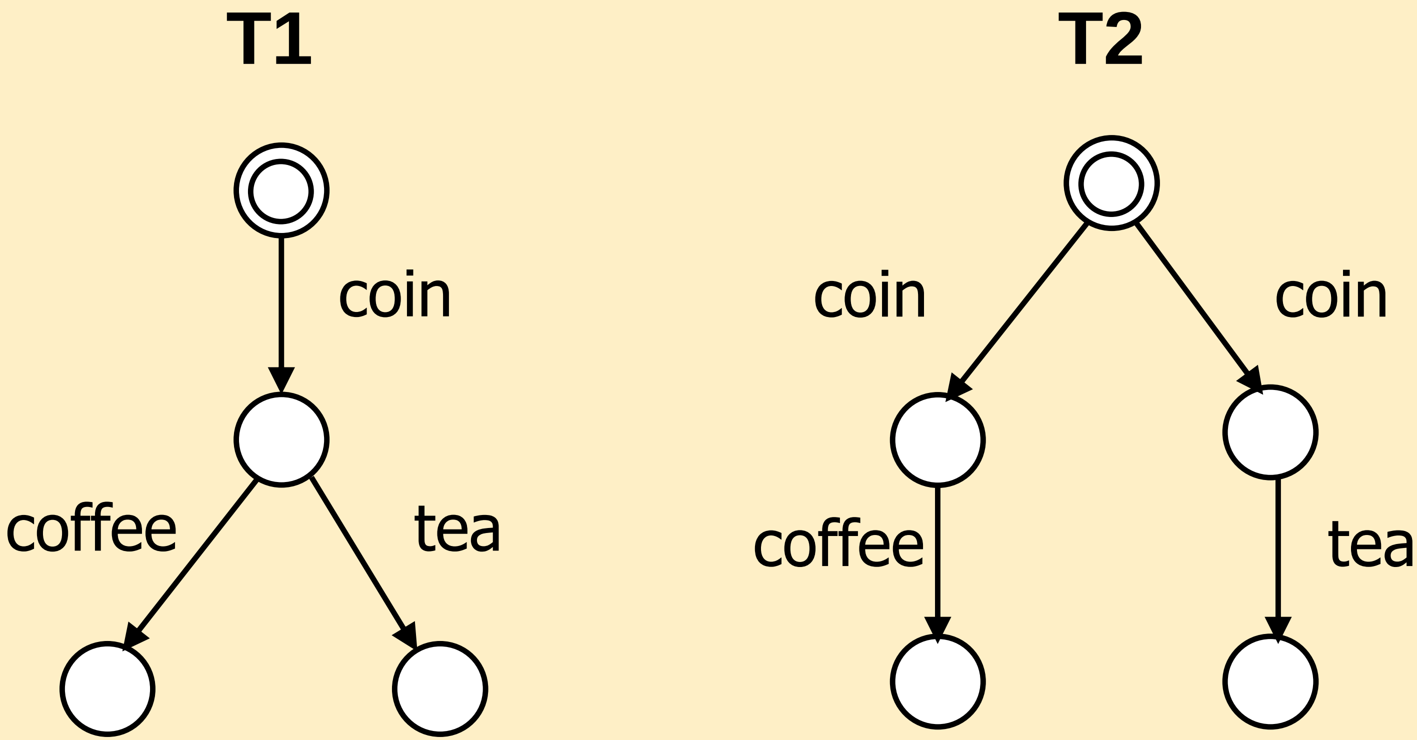
- $S = \{s_1, s_2, \dots, s_n\}$ is a finite set of states,
- $I \subseteq S$ is the set of initial states,
- $\rightarrow \subseteq S \times Act \times S$ is the set of transitions

We denote by $s \overset{a}{\rightarrow} s'$ iff $(s, a, s') \in \rightarrow$.

Example for LTS

Vending machine

- $Act = \{coin, coffe, tea\}$



Kripke Transition System

- Expresses properties of both states and transitions: labeling by atomic propositions and actions
- Possibly more than one labels per state, exactly one action per transition

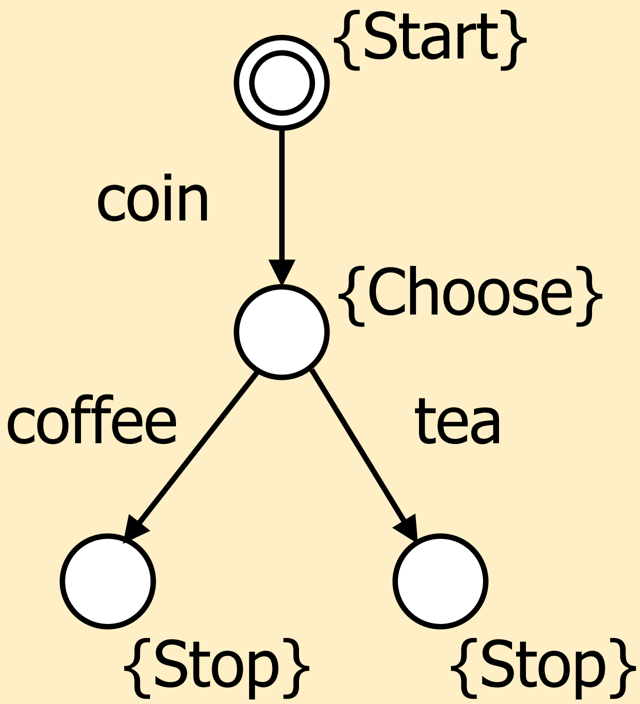
A Kripke transition system KTS over a set of atomic propositions AP and set of actions Act is a tuple $(S, I, \rightarrow, L)$ where

- $(S, I, \rightarrow)$ is an LTS
- $L : S \rightarrow 2^{AP}$ is the labeling of states by atomic propositions

Example for KTS

Vending machine with state labeling

- $Act = \{\text{coin}, \text{coffee}, \text{tea}\}$
- $AP = \{\text{Start}, \text{Choose}, \text{Stop}\}$



Timed Automata
and the UPPAAL Model Checker

Automata and variables

- Goal: modeling state based behavior
- Basic formalism: finite state automaton (FSA)
  - Locations (named)
  - Edges
- Language extension: integer variables
  - Variables with restricted domain (e.g. int[0, 1] id)
  - Constants
  - Integer arithmetic
- Use: on transitions
  - Guard: predicate over variables
    - The transition can fire only if predicate holds
  - Action: variable assignment

Extension with clock variables

- Goal: modeling real-time behavior
  - Time passes in locations
  - Relative measurment of time (e.g. time-out): resetting and reading clock variable
  - Time dependent behavior
  - Property to check: set of reachable locations within time bound
- Language extension: clock variables
  - Measure time elapse by a constant rate
- Use: on transitions
  - Guard: predicate over clock variables
  - Action: resetting clocks to zero
- Use: on locations
  - Location invariant: predicate over clock variables, restricts time elapse for current location

Timed automata in UPPAAL

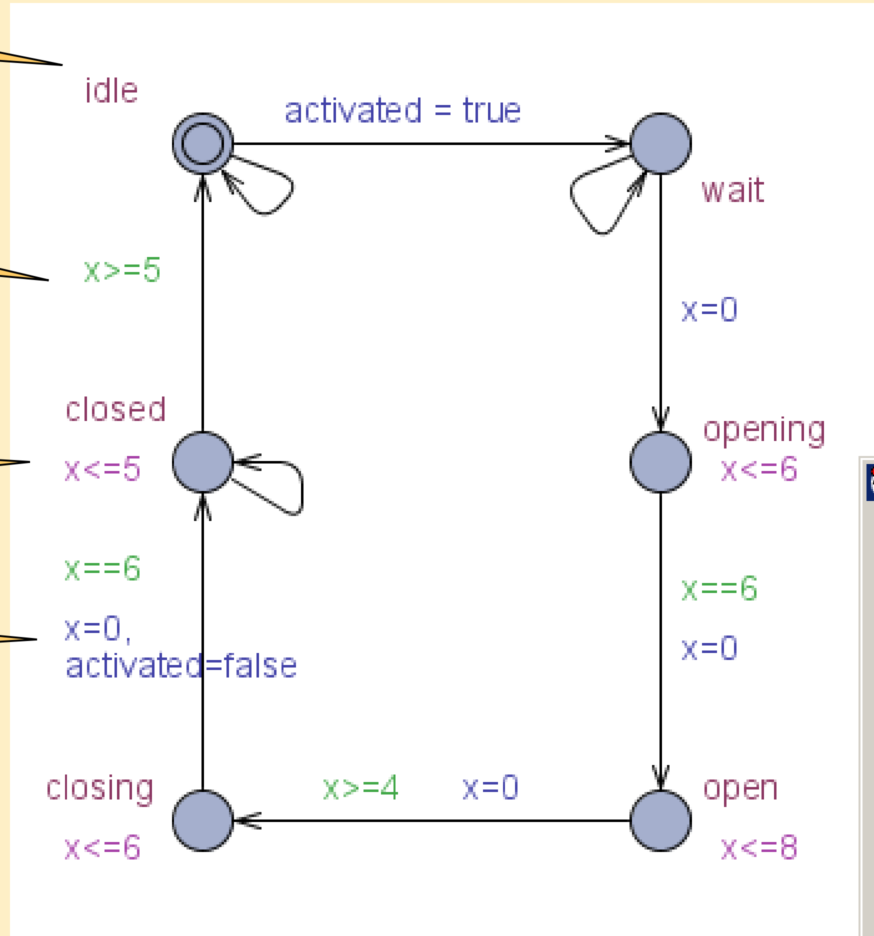
clock x;

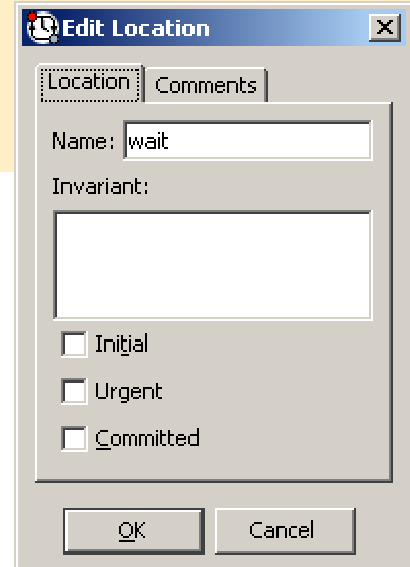
Location

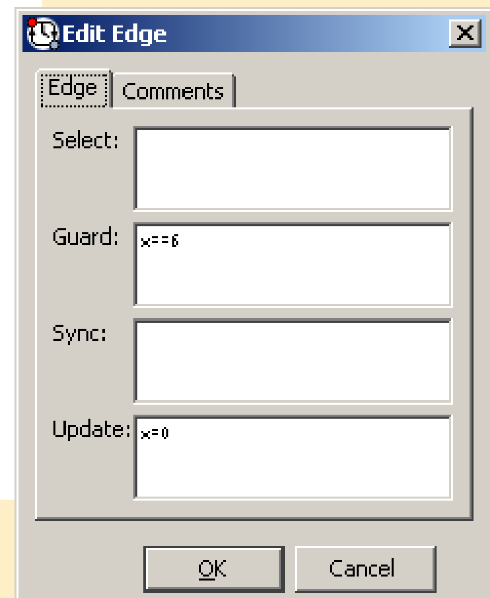
Guard

Invariant

Action

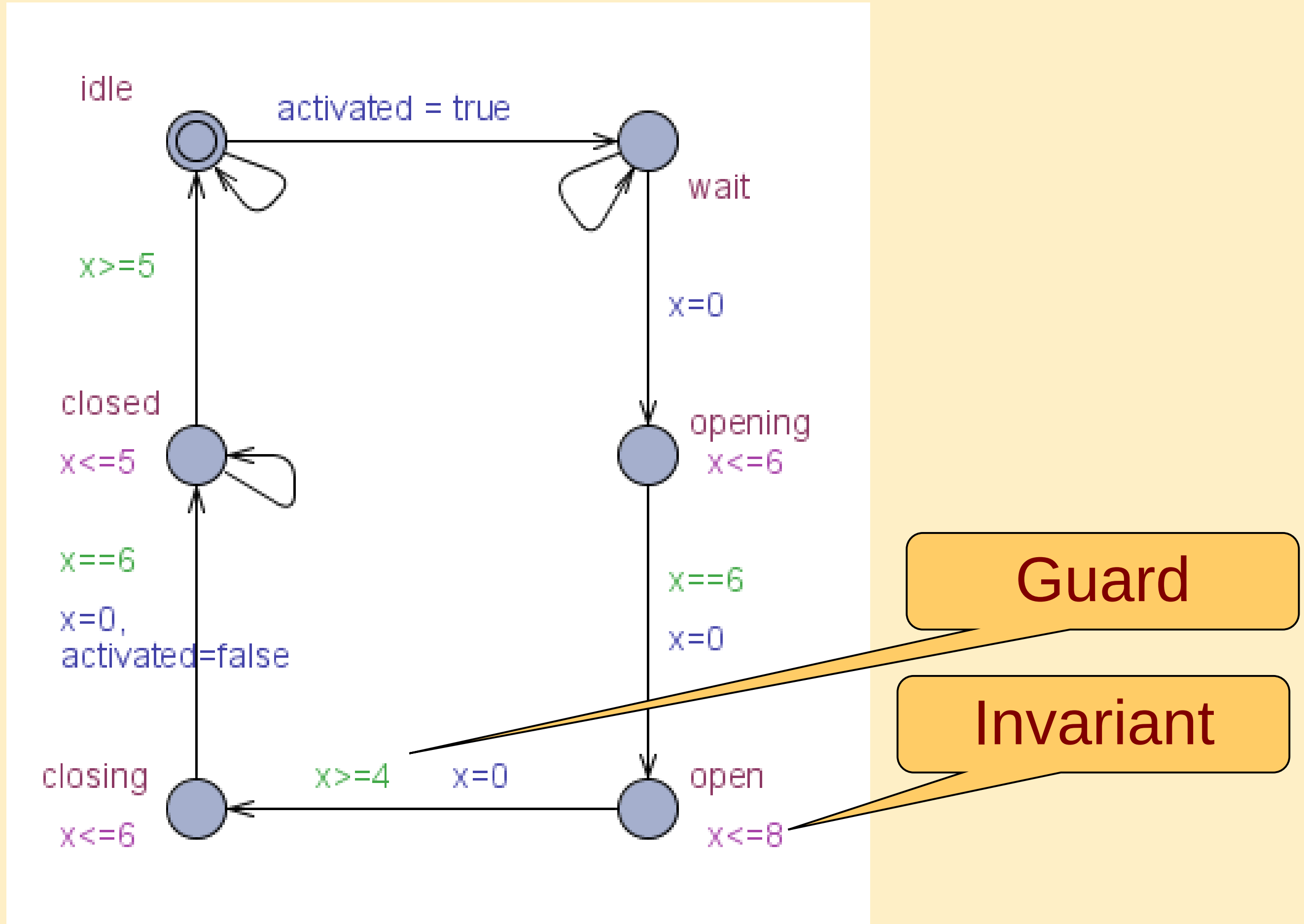






Role of guards and invariants

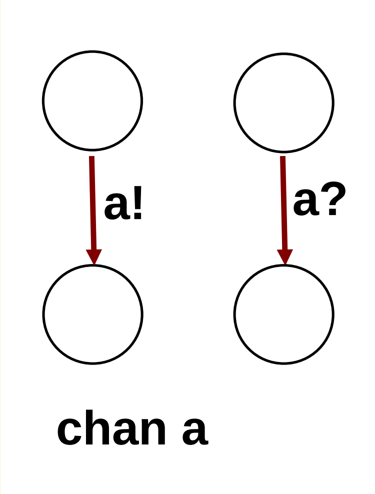
clock x;



Upon exiting location open, the value of clock is in interval [4, 8]

Extensions for concurrency

- Goal: modeling networks of automata
  - Synchronization between automata
  - Synchronized transitions (handshake):
    - Sending and receiving a message occurs at the same time
    - Enables modeling of asynchronous behavior as well
- Language extension: synchronized actions
  - Channels
  - Sending a message: ! operator
    Receiving a message: ? operator
    - E.g.: synchronization labels a! and a? for channel a
- Parameterization
  - Parameterized channels: arrays of channels
    - E.g. channel a[id] for a variable id
  - Parameterized automata: instantiating templates
    - E.g. automaton Door(true) for template Door(bool id)



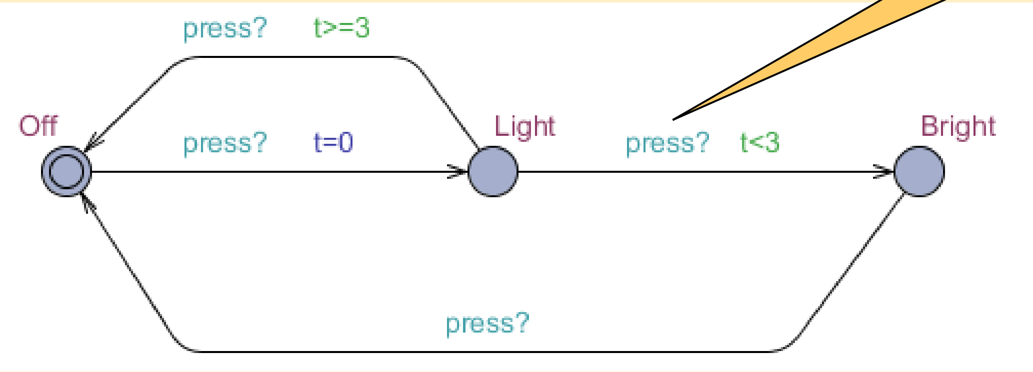
Example for clocks and synchronization

Declarations:

    clock t, u;
    chan press;

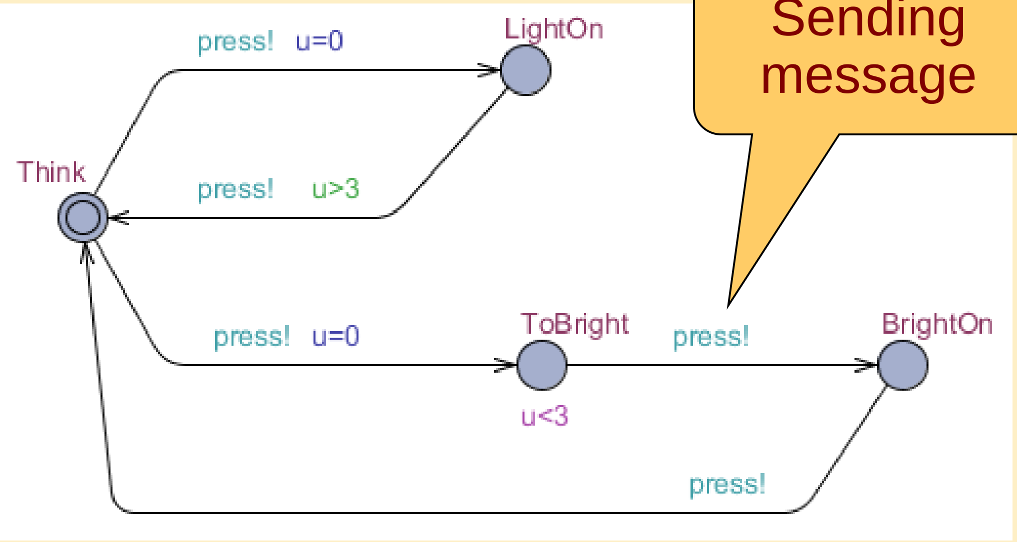
Switch:

Receiving message



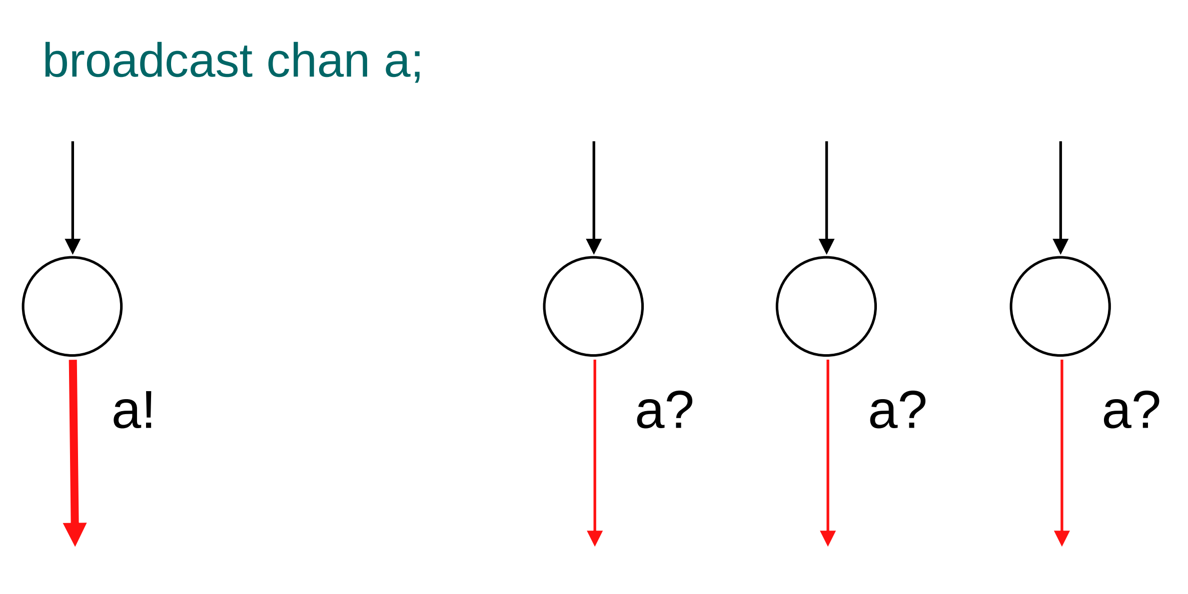
User:

Sending message



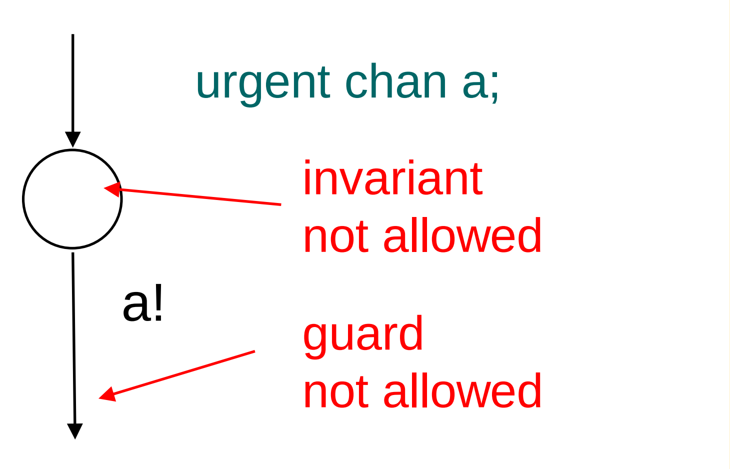
Further extensions: broadcast channel

- Broadcast channel: one-to-many communication
  - Sending message without condition
    - No handshake needed
  - All processes ready to receive message will synchronize
    - Receiving edge can only be taken upon receiving message
  - Restriction: no guard on receiving edge



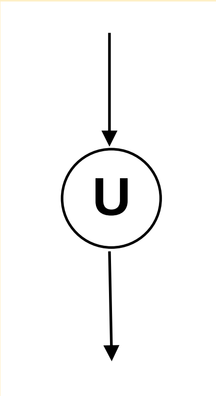
Further extensions: Urgent channel

- Urgent channel: prohibit time delay
  - The synchronization is executed without delay, (other edges might be traversed before, but only instantly)
  - Restrictions:
    - No guard is allowed on an edge labeled with the name of an urgent channel
    - No invariant is allowed on a location that is the source of an edge labeled with the name of an urgent channel

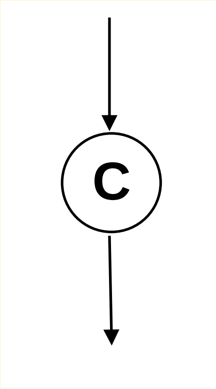


Further extensions: special locations

- Urgent location: prohibit time delay
  - Time is not allowed to progress in the location
  - Equivalent model:
    - Introuduce a clock variable: clock x;
    - Reset clock on all incoming edges: x:=0
    - Add invariant: x<=0



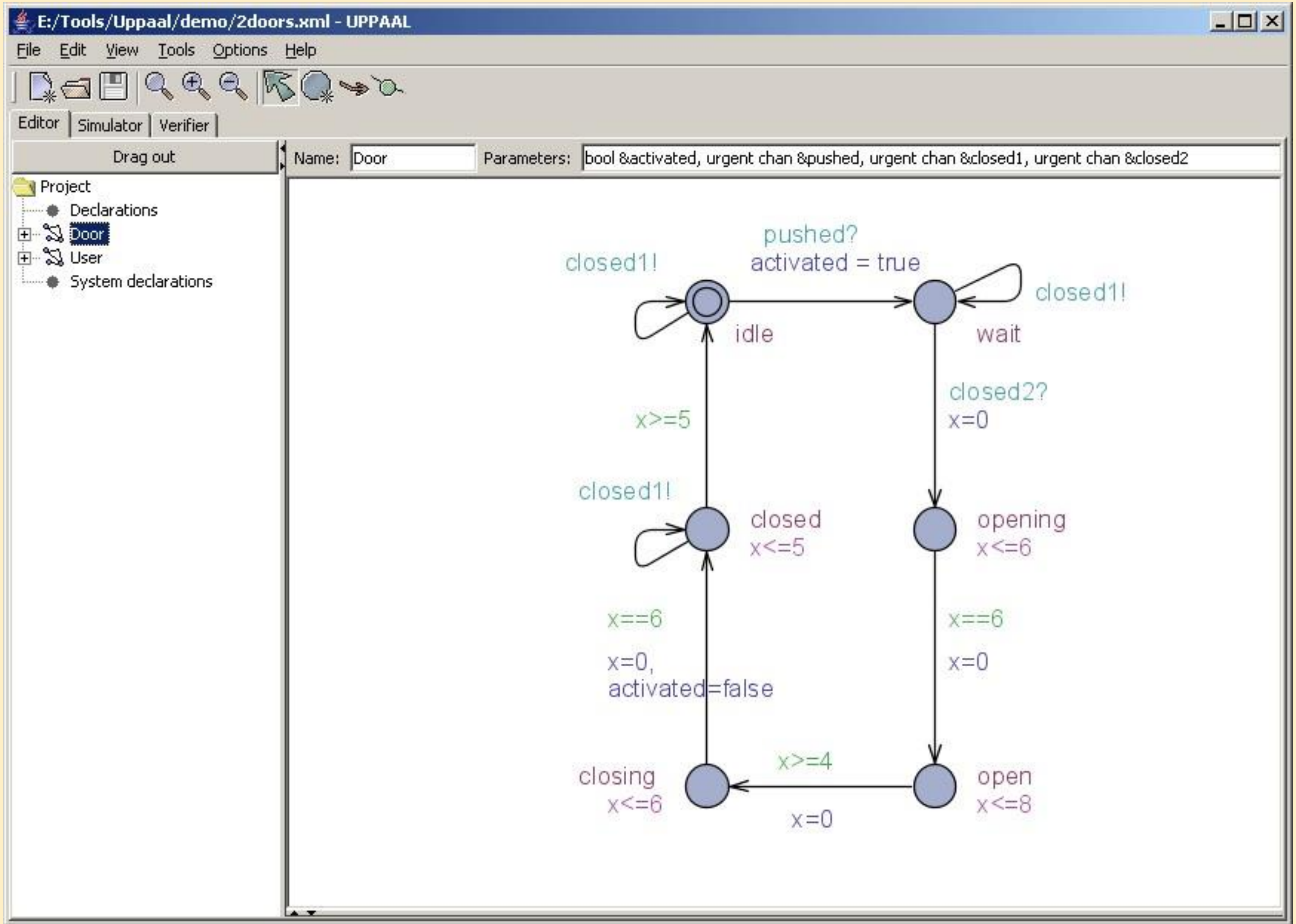
- Committed location: even more restrictive
  - A committed location is urgent
  - Committed state: at least one committed location is active
  - The next transition from a committed state must involve at least one out-edge of an active committed location



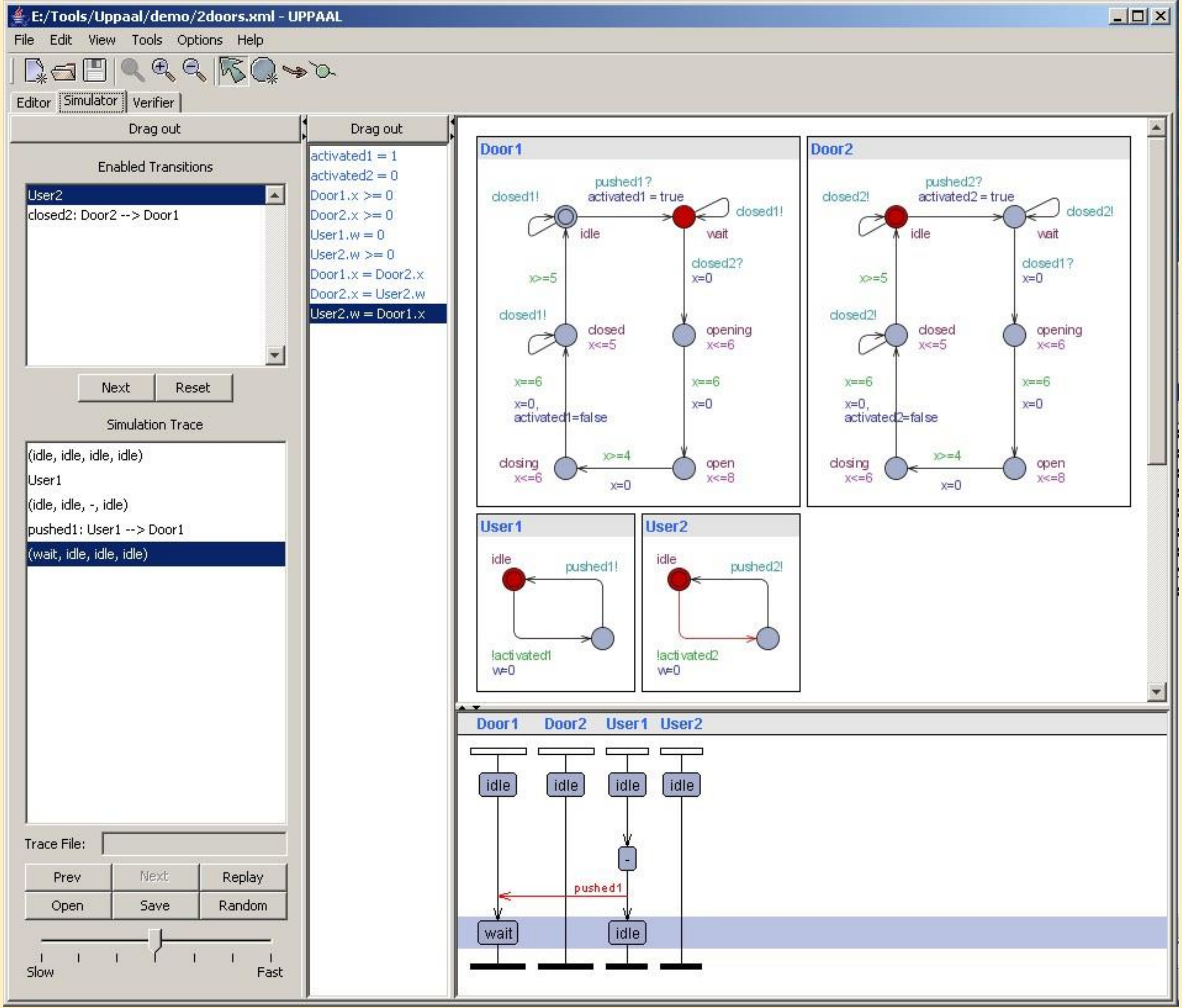
The UPPAAL model checker

- Development (1999-):
  - Uppsala University, Sweden
  - Aalborg University, Denmark
- Web page (information, examples, download):
  http://www.uppaal.org/
- Related tools:
  - UPPAAL CoVer: Test generation
  - UPPAAL TRON: On-line testing
  - UPPAAL PORT: Component based modeling
  - …
- Commercial version:
  http://www.uppaal.com/

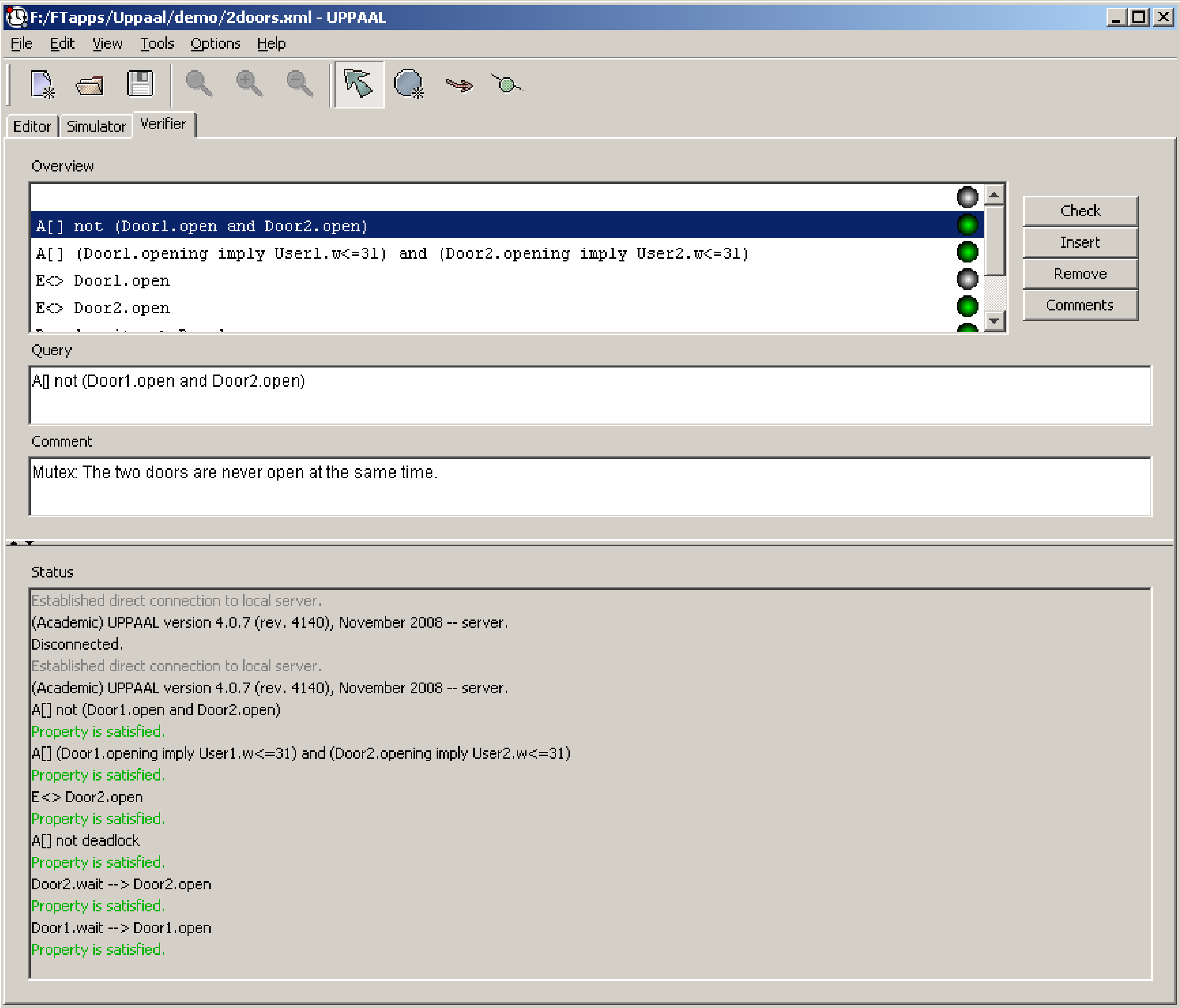
Automaton model



Simulator


E:/Tools/Uppaal/demo/2doors.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator Verifier
Drag out
Enabled Transitions
User2
closed2: Door2 --> Door1
Next Reset
Simulation Trace
(idle, idle, idle, idle)
User1
(idle, idle, -, idle)
pushed1: User1 --> Door1
(wait, idle, idle, idle)
Trace File:
Prev Next Replay
Open Save Random
Slow Fast
Drag out
activated1 = 1
activated2 = 0
Door1.x >= 0
Door2.x >= 0
User1.w = 0
User2.w >= 0
Door1.x = Door2.x
Door2.x = User2.w
User2.w = Door1.x
Door1
pushed1?
activated1 = true
closed1!
idle
wait
closed2?
x=0
x>=5
closed
x<=5
opening
x<=6
x==6
x=0,
activated1=false
closing
x<=6
x>=4
open
x<=8
Door2
pushed2?
activated2 = true
closed2!
closed1?
activated2=false
User1
idle
pushed1!
!activated1
w=0
User2
pushed2!
!activated2
Door1 Door2 User1 User2
idle
-
pushed1
wait

Verification



Motivating example
(optional)

Motivating exampler: mutual exclusion

- 2 processes, 3 shared variables (H. Hyman, 1966)
  - **blocked0**: process 1 (P0) wants to enter
  - **blocked1**: process 2 (P1) wants to enter
  - **turn**: which process is allowed to enter (0 for P0, 1 for P1)

P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

P1

```
while (true) {
    blocked1 = true;
    while (turn!=1) {
        while (blocked0==true) {
            skip;
        }
        turn=1;
    }
    // Critical section
    blocked1 = false;
    // Do other things
}
```

Is the algorithm correct?

The model in UPPAAL (version 1)

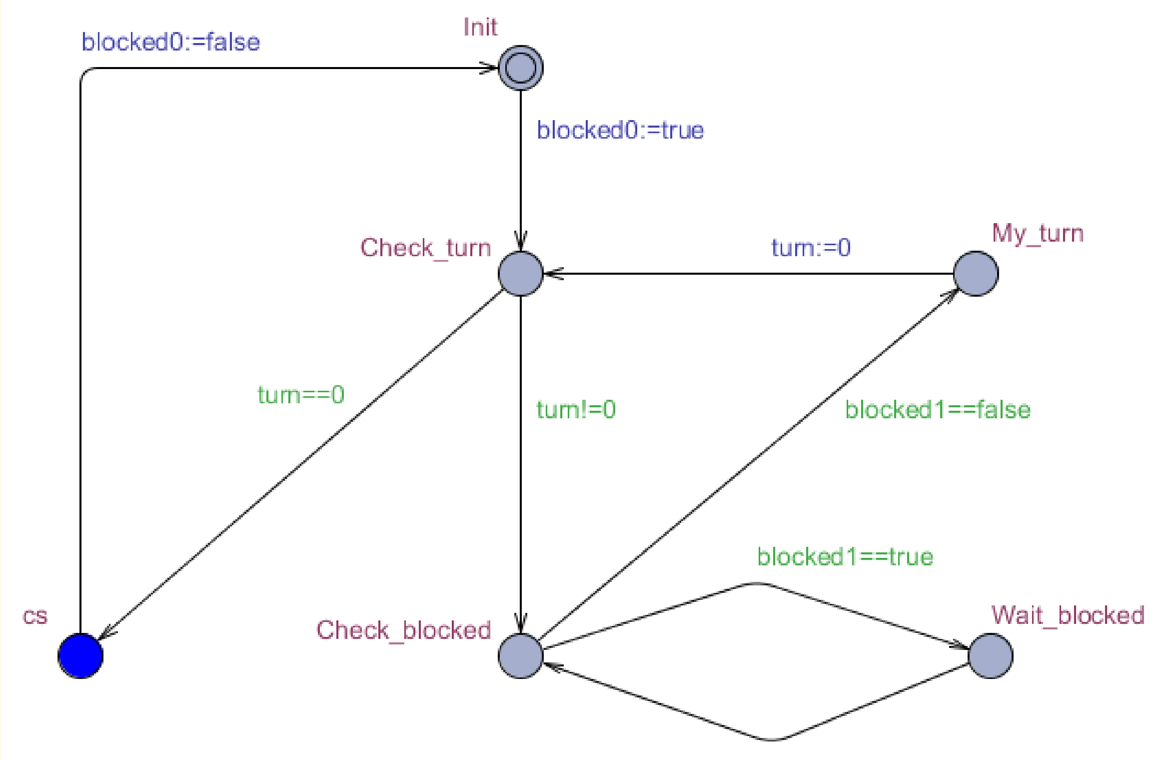
Declarations:

    bool blocked0;
    bool blocked1;
    int[0,1] turn=0;
    system P0, P1;

Used modeling idioms:

- Global variables
- Variables with restricted domain

Automaton P0:



P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

The model in UPPAAL (version 2)

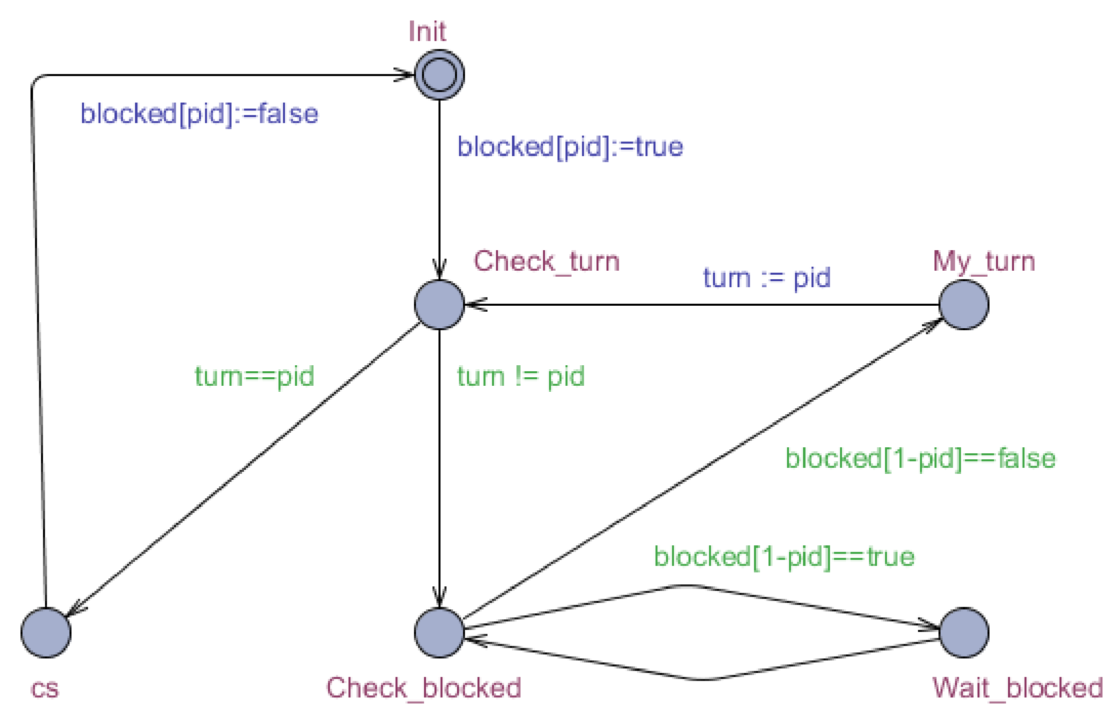
Declarations:

```
bool blocked[2];
int[0,1] turn;
P0 = P(0);
P1 = P(1);
system P0,P1;
```

Template P
with parameter pid:

Used modeling idioms:

- Global variables
- Variables with restricted domain
- Modeling common behavior with templates
- Template instantiation with parameters
- Variables of array type



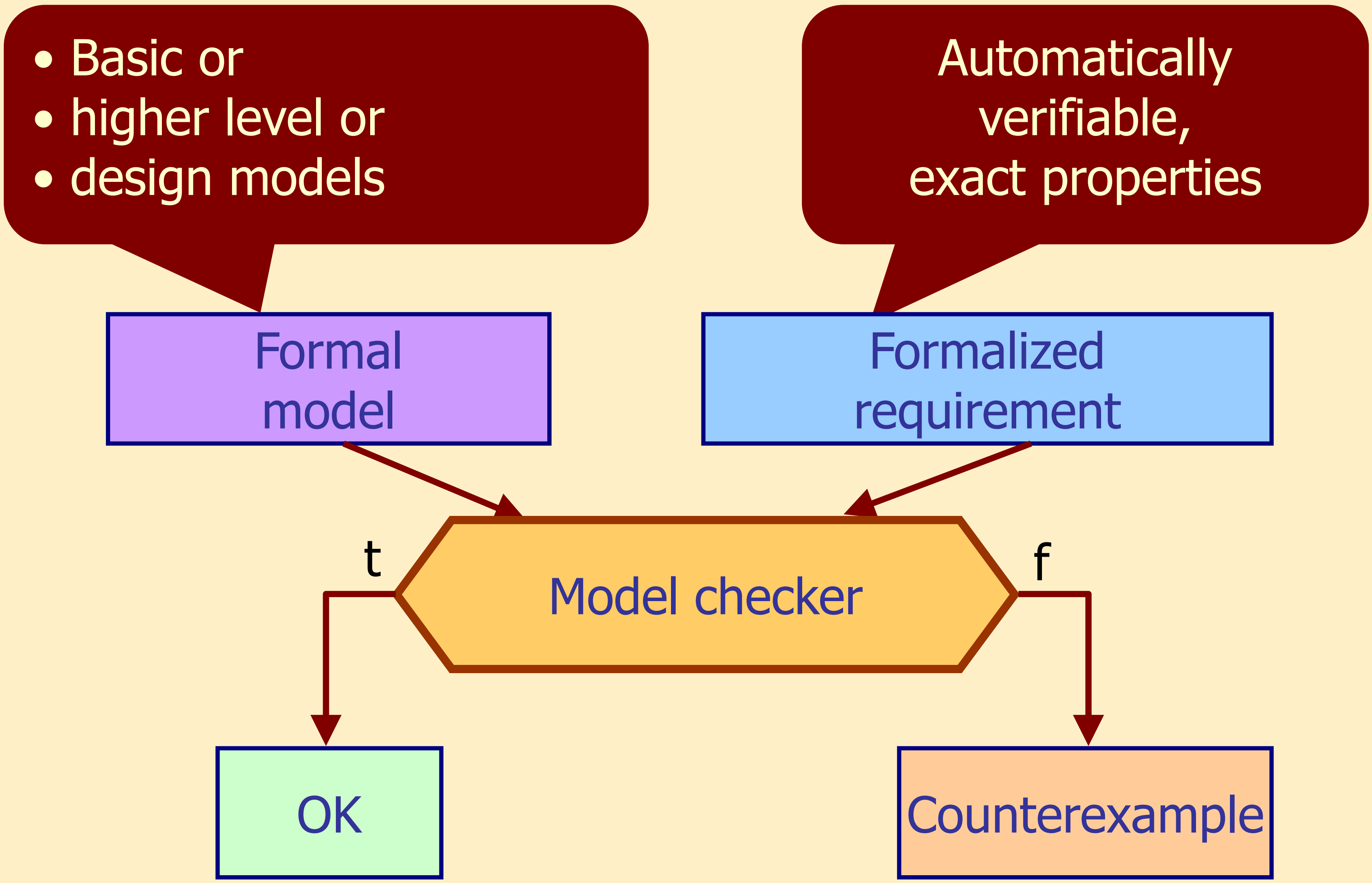
P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

Properties to verifiy

- Mutual exclusion:
  - At most one process is allowed to be in the critical section
- The expected behavior is possible:
  - For P0 it is possible to enter the critical section
  - For P1 it is possible to enter the critical section
- Starvation freedom:
  - P0 will eventually enter the critical section
  - P1 will eventually enter the critical section
- Deadlock freedom:
  - It is not possible that processes are mutually waiting for each other

Our goal


• Basic or
• higher level or
• design models
Automatically verifiable, exact properties
Formal model
Formalized requirement
Model checker
t
f
OK
Counterexample