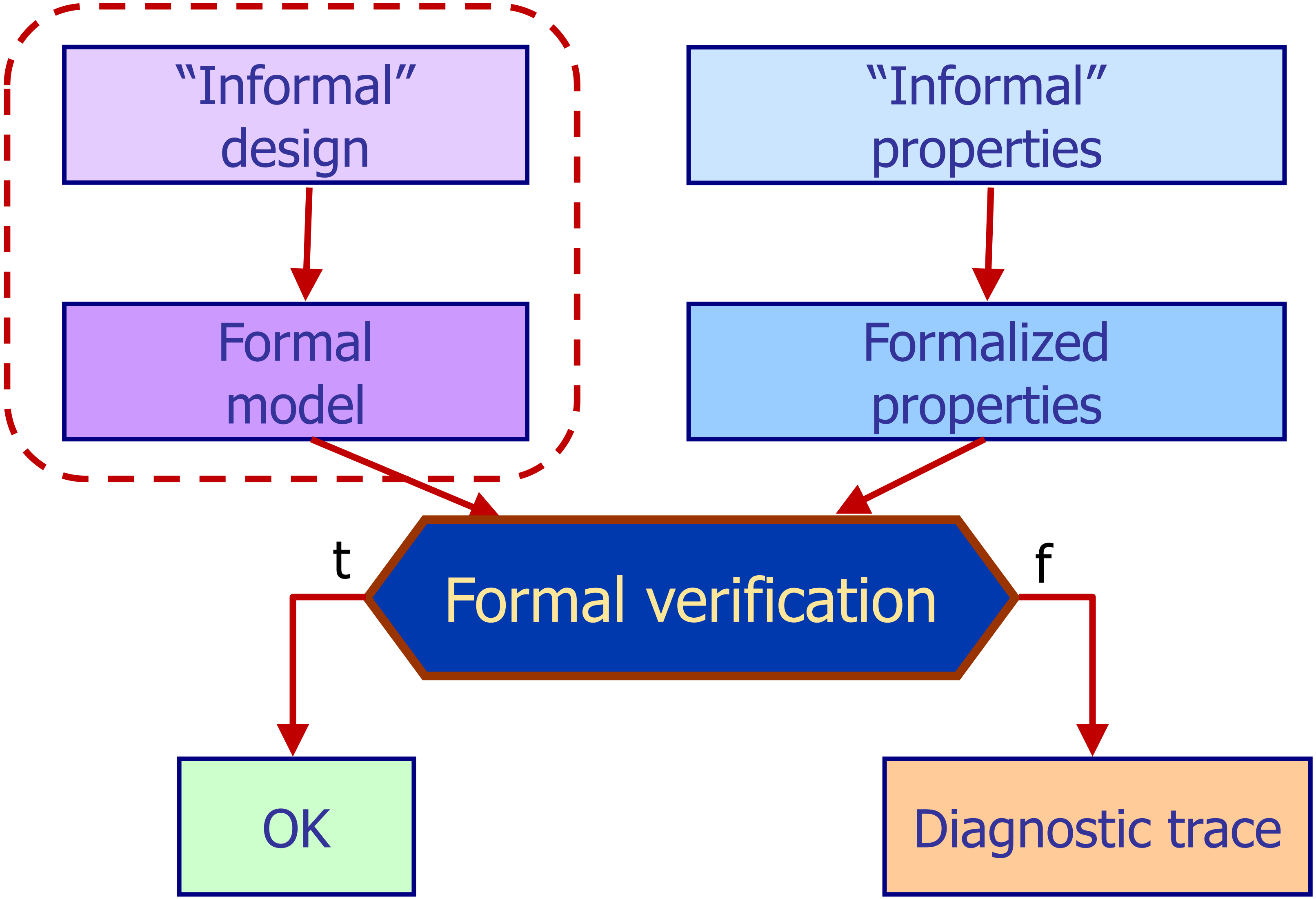
Formal verification: Basic formalisms

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Recap: The goal of formal verification



Basic structures

Kripke structure (KS)

Labeled transition system (LTS)

Kripke transition system (KTS)

Finite state automata (FSA)

1. Kripke structure

Basic characteristics:

- Capturing properties of states: labeling by atomic propositions
- Possibly more than one labels per state
- Goal: characterizing the behavior through labeled states

Definition:

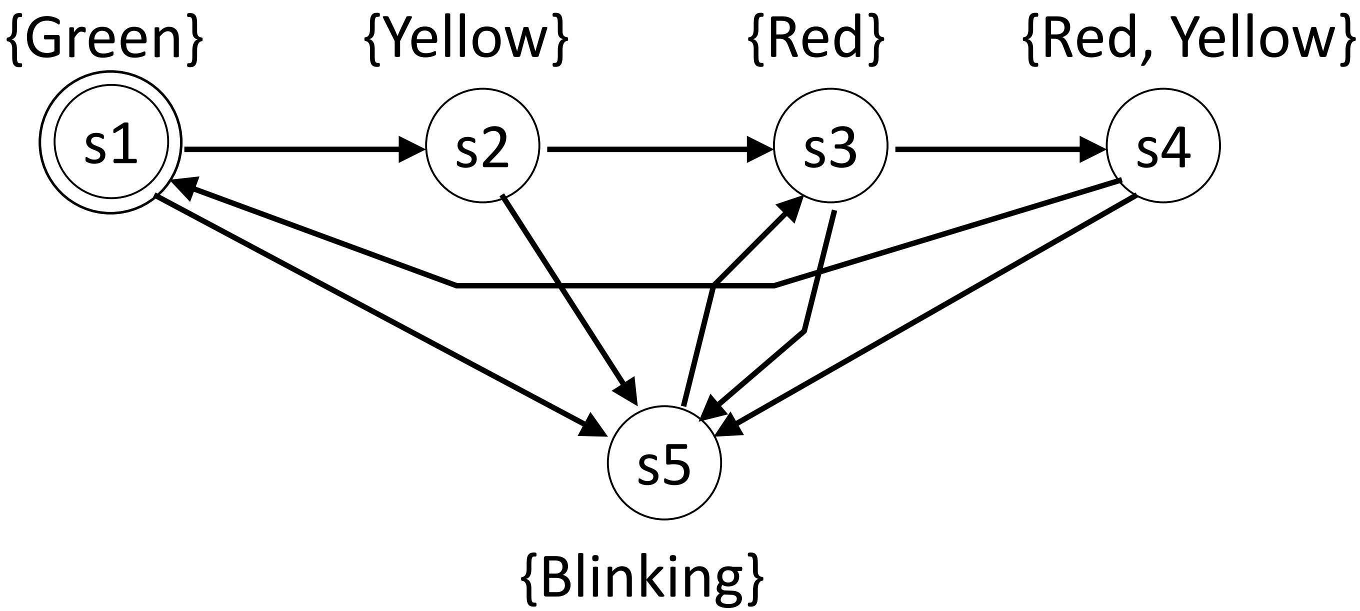
A Kripke structure KS over a set of atomic propositions $AP = \{P, Q, R, \dots\}$ is a tuple (S, R, L) where

- $S = \{s_1, s_2, \dots, s_n\}$ 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: Kripke structure

Traffic light controller

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



2. Labeled transition system

Basic characteristics:

- Capturing properties of transitions: labeling by actions
- Exactly one action per transition
- Goal: characterizing communication and protocols by actions

Definition:

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

- $S = \{s_1, s_2, \dots, s_n\}$ is a finite set of states,

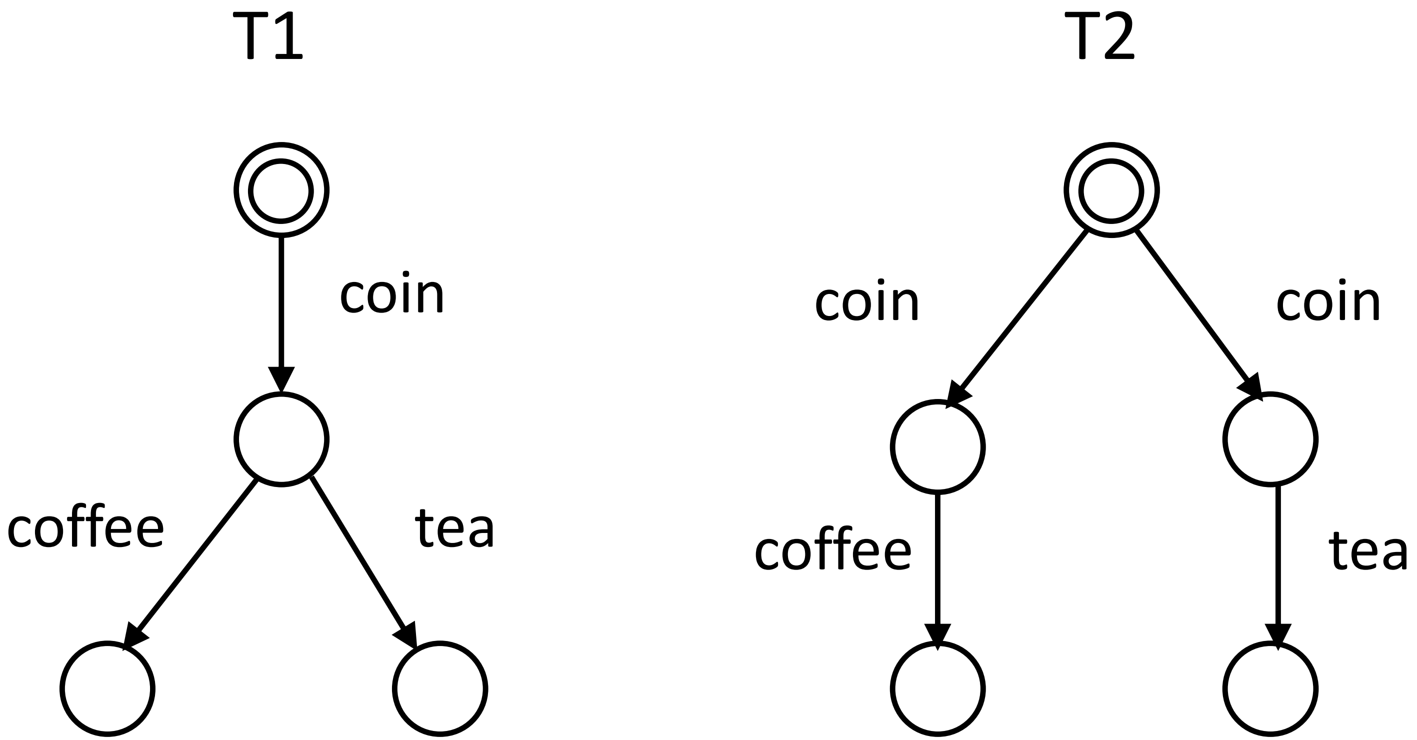
  $I \subseteq S$ is the set of initial states,
- $\to : S \times Act \times S$ is the set of transitions

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

Example: Labeled transition system

Vending machine

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



3. Kripke transition system

Basic characteristics:

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

Definition:

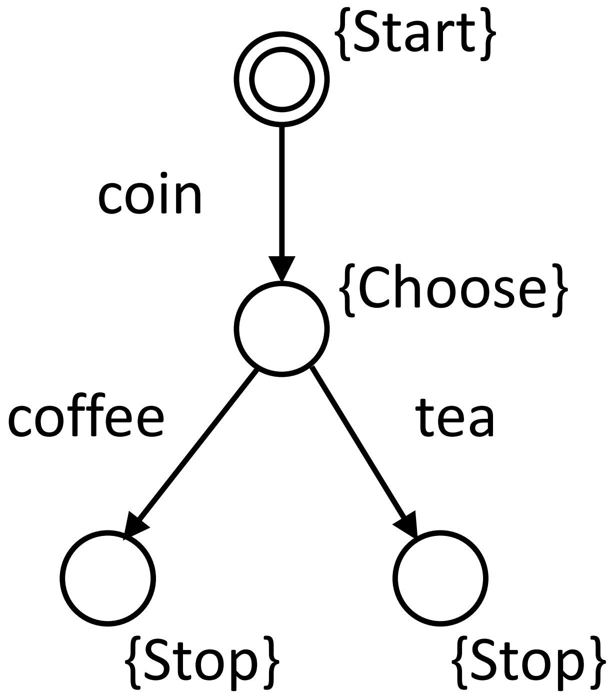
A Kripke transition system KTS over a set of atomic propositions AP and set of actions Act is a tuple $(S, \rightarrow, L)$ where

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

Example: Kripke transition system

Vending machine with state labeling

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



4. Automata on finite words

- $A=(\Sigma, S, S_0, \rho, F)$ where
  - Σ alphabet, S states, S_0 initial states
  - ρ state transition relation, $\rho: S \times \Sigma \to 2^S$
  - F set of accepting states
- Run of an automaton
  - State sequence $r=(s_0, s_1, s_2, \dots s_n)$ on the incoming word $w=(a_0, a_1, a_2, \dots a_n)$
  - r is an accepting run if $s_n \in F$
  - A word w is accepted by the automaton, if there is an accepting run over w
- Language L accepted by the automaton A:

$$L(A)=\{ w \in \Sigma^* \mid w \text{ accepted}\}$$

Automata on infinite words

- Infinite words
  - Accepting state at the end of a word cannot be checked
- Büchi acceptance criterion:
  - On the incoming infinite word $w=(a_0, a_1, a_2, \dots)$ there is an $r=(s_0, s_1, s_2, \dots)$ infinite state sequence
  - $lim(r)=\{s \mid s$ occurs infinitely many times, i.e., there is no j, such that $\forall k>j: s \neq s_k\}$
  - Accepting run: $lim(r) \cap F \neq 0$
  - A word w is accepted by the automaton, if there is an accepting run over w (i.e., accepting state occurs infinitely often along w)
- Language L accepted by the automaton A:

$$L(A)=\{ w \in \Sigma^* \mid w \text{ accepted}\}$$

Timed Automata: Finite State Automata with Time

Timed Automata in the UPPAAL model checker

Timed Automata: Extension with variables

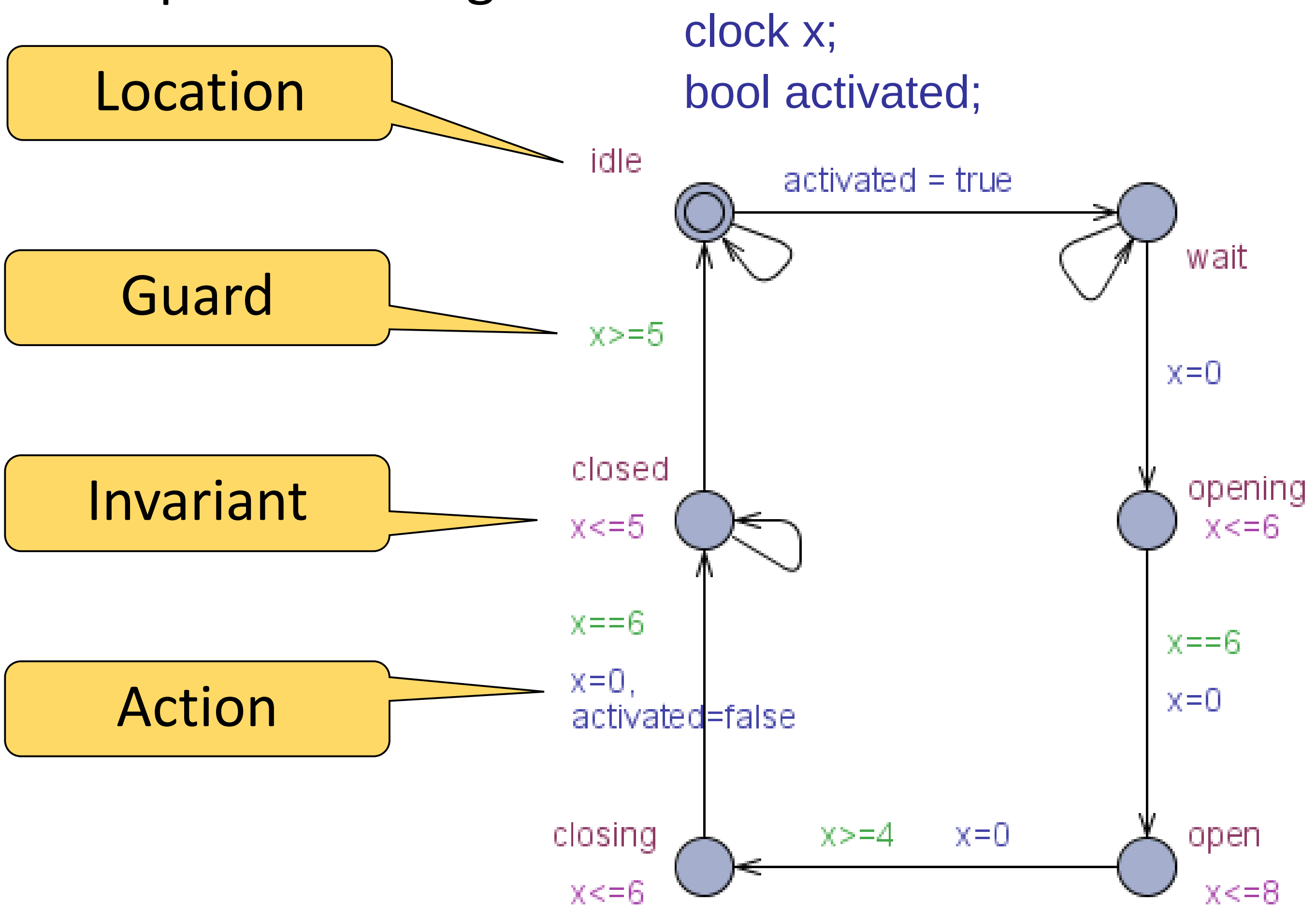
- Basic formalism: Finite state automaton (FSA)
  - Control locations (named) – part of the state of the automaton
  - Edges – define state transitions
- Language extension: integer variables
  - Variables with restricted domain (e.g. int[0, 10] id)
  - Constants (e.g., const int N = 6)
  - Integer arithmetic
- Use of variables: on edges
  - Guard: predicate over variables
    - The state transition can occur only if the predicate holds
  - Action: variable assignment

Timed automata: Extension with clock variables

- Goal: modeling time-dependent behavior
  - Time passes in given states of the component
  - Relative time measurement by resetting and reading timers; behavior depends on timer value (e.g., timeout)
- Language extension: clock variables
  - Measuring time elapse by a constant rate
- Use of clock variables on edges
  - Guard: predicate over clock variables
  - Action: resetting clocks to zero
- Use of clock variables on locations
  - Location invariant: predicate over clock variables; being in a location is valid until its invariant holds

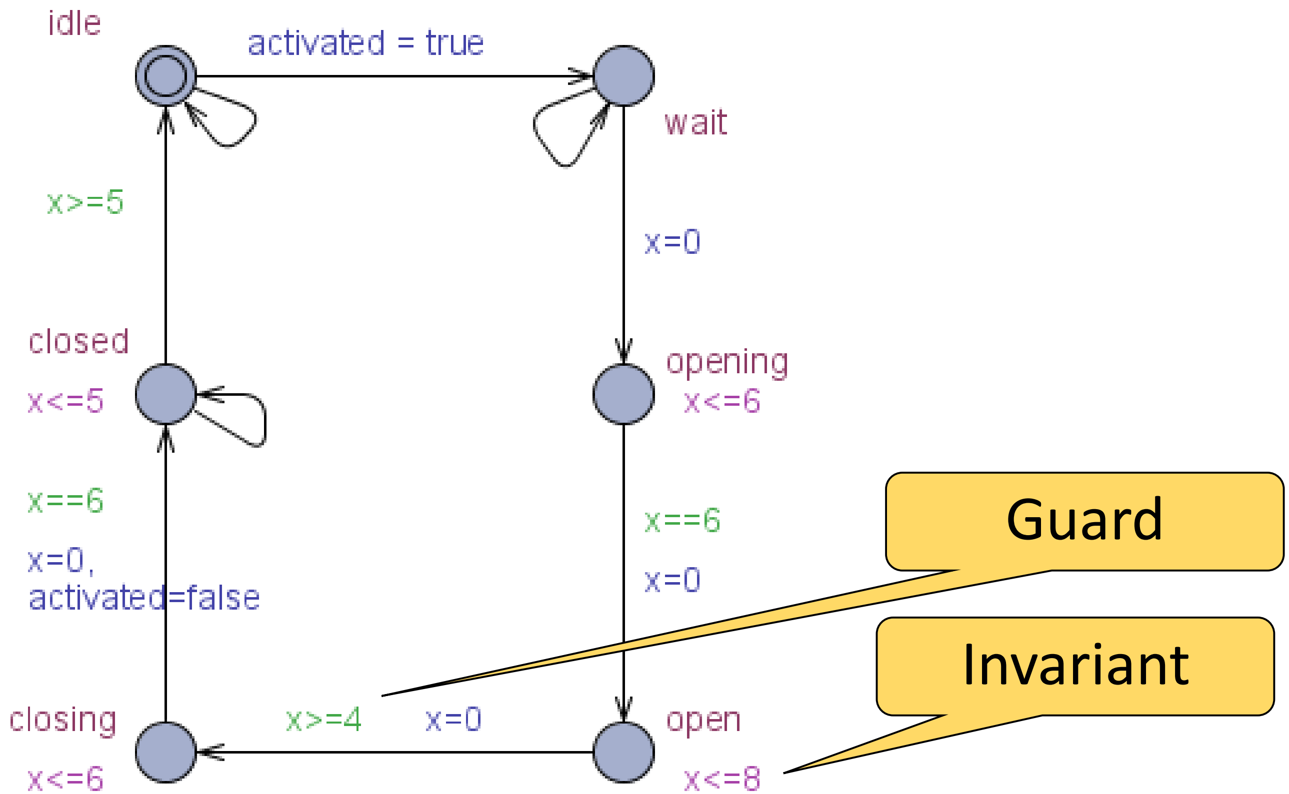
Timed automata in UPPAAL

Example: revolving door



Role of guards and invariants

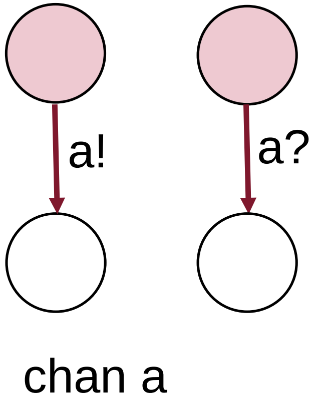
clock x;



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

Extensions for modeling concurrency

- Goal: modeling networks of automata
  - Interaction: Synchronization between automata transitions
  - Synchronous communication (handshake, rendezvous)
    - Sending and receiving a message occurs at the same time
    - Modeling of asynchronous behavior is possible by modeling channels
- Language extension: synchronized actions
  - Declaring channels for sending messages
  - Sending a message: ! operator
    Receiving a message: ? operator
  - E.g.: synchronization labels a! and a? for channel a



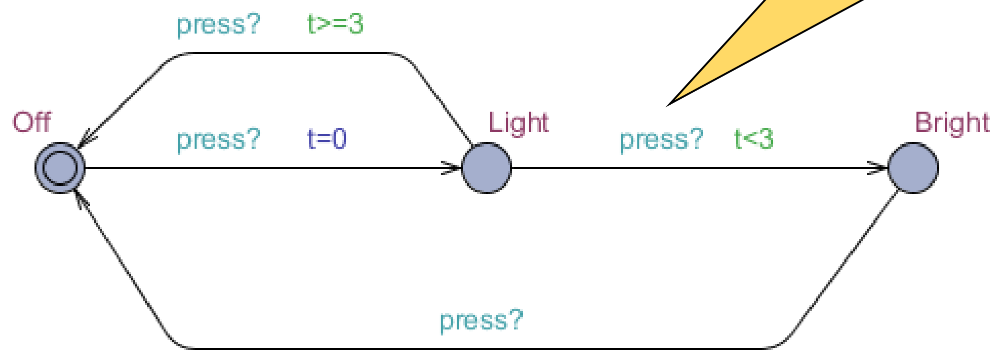
- Parameterization
  - Arrays of channels: E.g. channel a[id] for a variable id
  - Useful in case of several participants and interactions

Example for clocks and synchronization

Declarations:

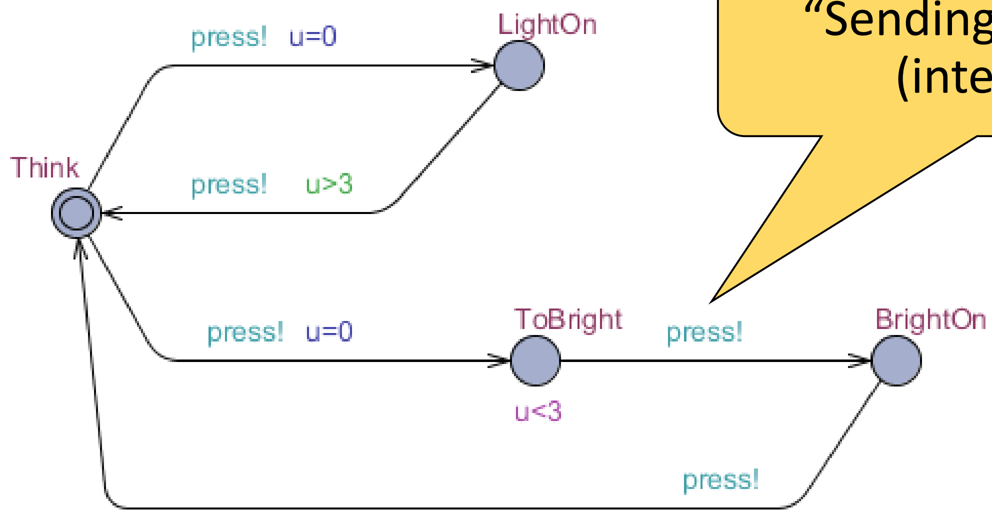
    clock t, u;
    chan press;

Switch:



“Receiving a message” (interaction)

User:

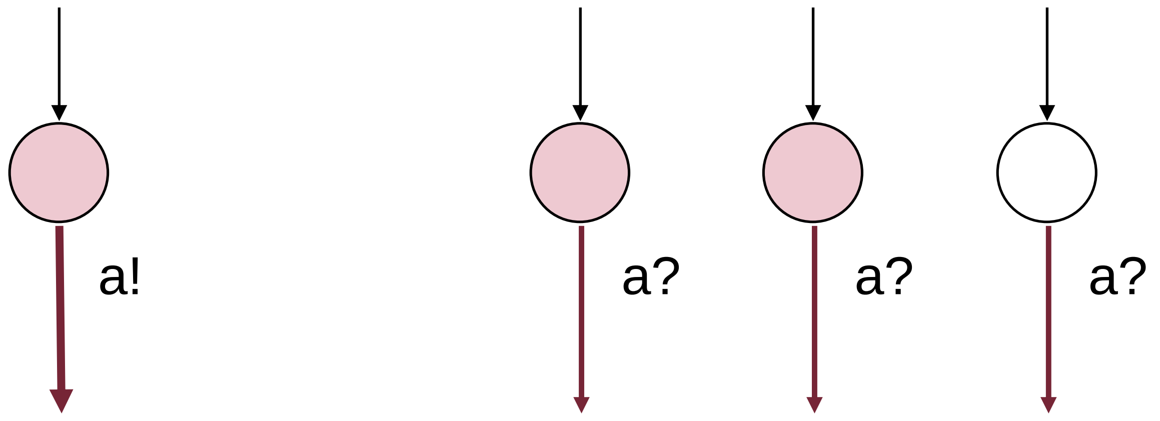


“Sending a message” (interaction)

Further extensions: broadcast channel

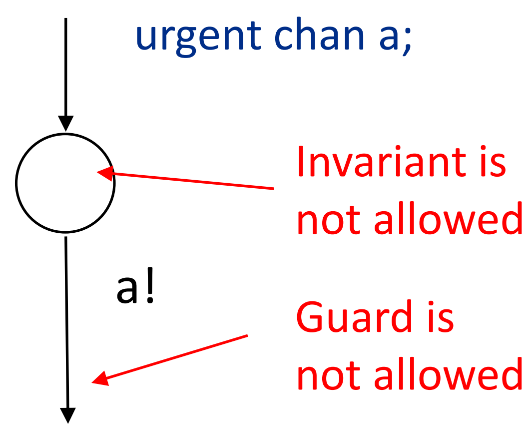
- Broadcast channel: one-to-many communication
  - Sending a message: unconditional
    - No handshake needed
  - Receiving a message: synchronized to the sender
    - All processes that are ready to receive the message will synchronize
  - Restriction: no guard on receiving edge

broadcast chan a;



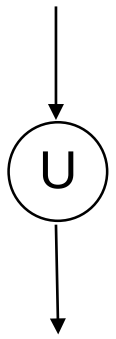
Further extensions: urgent channel

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

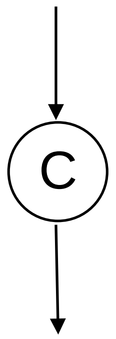


Further extensions: special locations

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



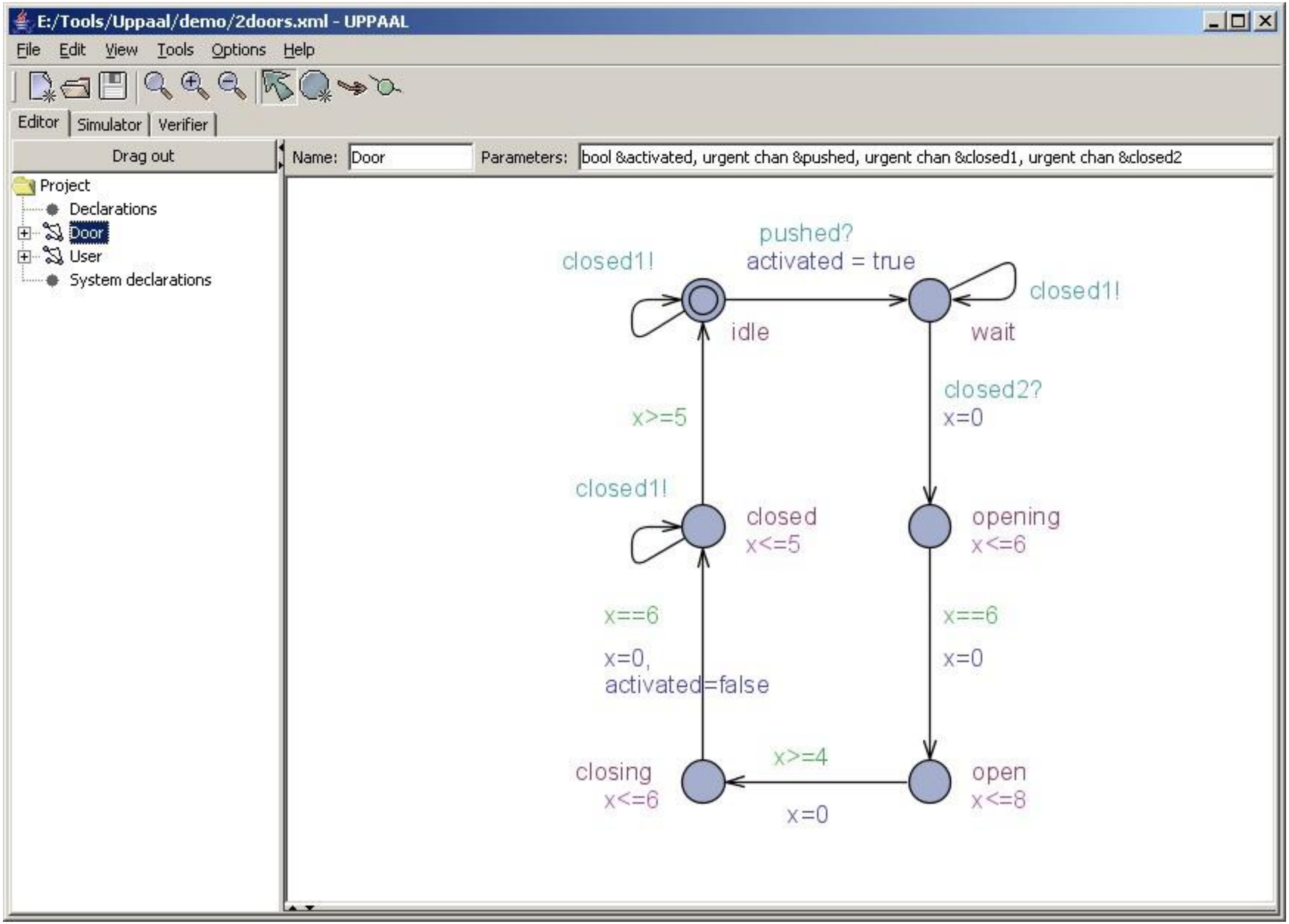
- Committed location: even more restrictive
  - A committed location is urgent
  - Committed state: at least one committed location is active
  - The next transition from a committed state must involve at least one out-edge of an active committed location
  - Simpler case: If only one committed location is active then its out-edge shall immediately follow its in-edge



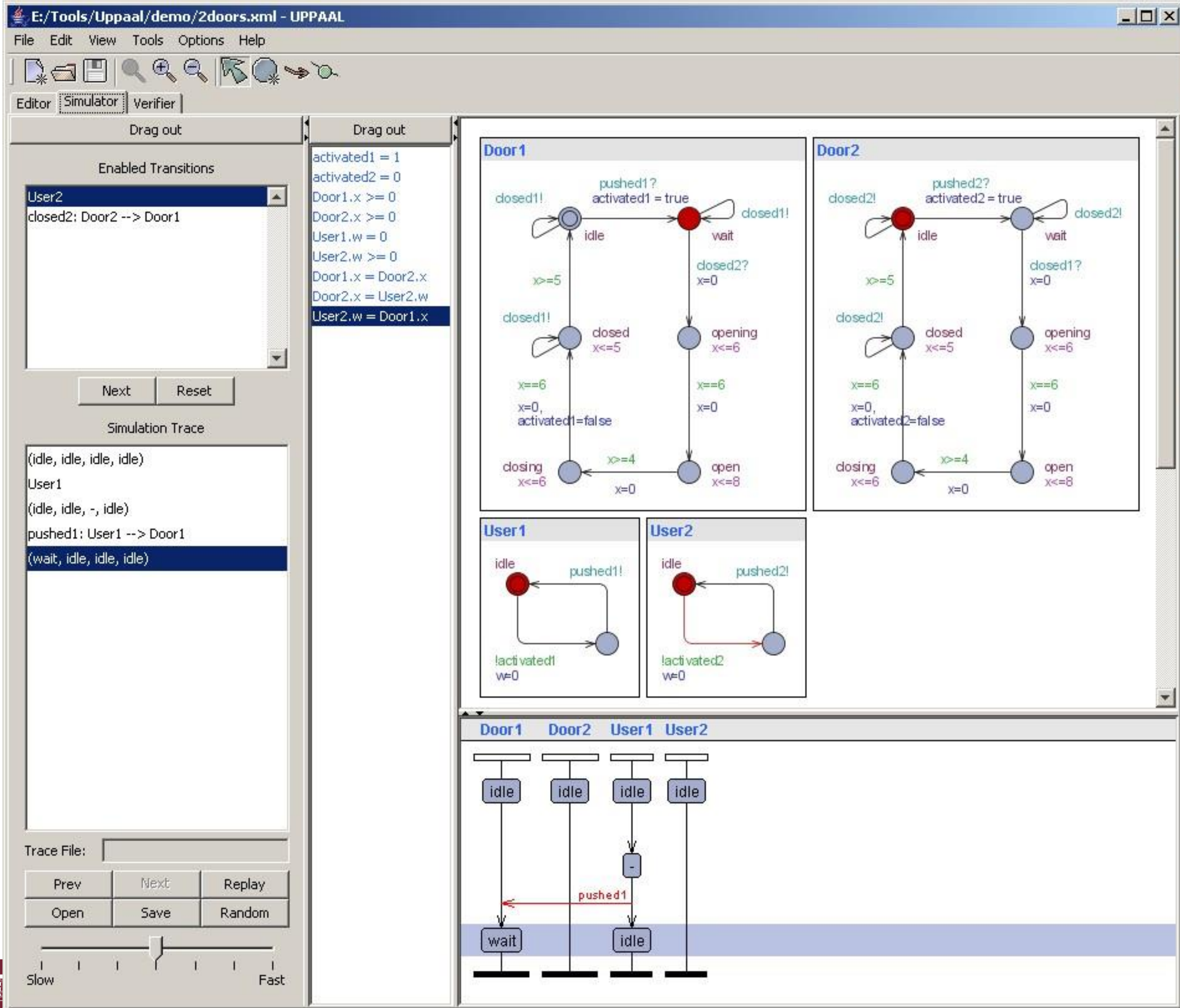
The UPPAAL model checker

- 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 idle idle idle
-
pushed1
wait idle

Verification