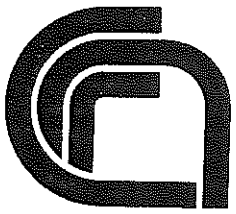


Consiglio Nazionale delle Ricerche

Introduction to UPPAAL
Slides

Mieke Massink

Technical Report
CNUCE-B04-2002-011

CNUCE

Pisa

Introduction to Model Checking

Mieke Massink

C.N.R.-Ist. CNUCE, Pisa

Consiglio Nazionale delle Ricerche, Pisa

Introduction to Model Checking – Mieke Massink – Feb. 2002

Overview

1. Introduction to model checking
2. Overview of the course:

The model checker SPIN

The model checker UPPAAL

Model Checking: An informal definition

Model checking is an *automated* technique that, given a:

finite state model of a system and

logical property

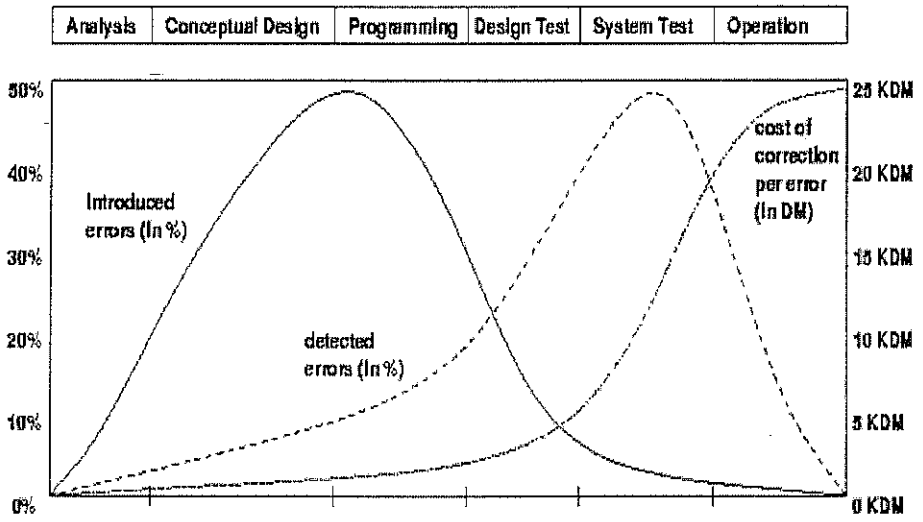
systematically checks whether this property holds for (a given initial state in) that model.

Formal Methods

Model Checking is one of four categories of system validation techniques based on Formal methods:

- Formal Verification: correctness proofs
- Model Checking
- Model Based (Symbolic) Simulation
- Model Based Testing (Formal Testing)

Introduction and detection of errors and relative costs.



Amounts in German Marks: 1 KDM is about 0.5 M lire. Source: Siemens, Liggesmeijer et al.

RISKS-Forum

Many problems related to risks involving use of computers are discussed at:

http://catless.ncl.ac.uk/Risks

Moderated by Peter G. Neumann

Introduction to UPPAAL2K

Mieke Massink

C.N.R.-Ist. CNUCE, Pisa

Consiglio Nazionale delle Ricerche, Pisa

Introduction to Model Checking – Mieke Massink – Feb. 2002

Overview

1. Introduction to real-time model checking
2. Global overview of UPPAAL2K
3. Model: Networks of Timed Automata
4. Symbolic Simulation of timed automata
5. Specification: Real-Time temporal logic properties
6. Verification of properties
7. Analysis of example traces
8. Case studies

Introduction to Real-Time Model Checking

Time-critical systems [Katoen, 99]

"Time-critical systems are those systems in which the correctness of their behaviour depends not only on the logical result of the computation but **also** on the **time** at which the results are produced"

Note: CTL, ACTL and LTL are *temporal logics* that focus on the **temporal order** of events.

Reasoning about time-critical systems requires that *quantitative* aspects of timing issues can be addressed.

For example: (LTL) $G[p => F\ q]$: Event A always followed by B
Missing: How much time passes between A and B?

Examples of time-critical systems

- *Train crossing.* Crossing needs to be closed before train passes.
- *Lip-synchronisation protocol.* Synchronises the separate video and audio sources into an understandable presentation. Bounds on amount of time passing between video frame presentation and related audio frame. Human tollerance of less than ca. 160 ms.
- *Bounded Retransmission Protocol.* Protocol developed by Philips for the communication of large files via infrared communication medium between a remote control unit and audio/video equipment. Its correctness depends critically on relationships between:
  - time-out values of timers at sender and receiver
  - transmission delays and
  - synchronisation delays

Quantitative time in temporal logics

Some of the issues that need to be addressed [Koymans 1989]

- Should time elements be represented explicitly or implicitly?
- Should the notion of time reference be absolute or relative?
- Should the time domain be continuous or discrete?

The debate on continuous vs. discrete time domains

Should the time domain be continuous or discrete?

- Newtonian Physics: time is continuous
- Computer systems interact with continuous environment
- Models include system and environment components

Modelling of *asynchronous* systems

- Computer systems are inherently discrete; they work in lock-step and processor cycles
- Computer systems sample their environment
- Time can be measured as number of steps; intermediate times are not useful

Modelling of *synchronous* systems

Some important real-time logics

- LTL-extensions
  - RTTL Real-time Temp. Logic, Wonham and Ostroff 1985
  - Metric Temporal Logic, Koymans, 1990
  - Explicit Clock Temp. Logic, Harel, Pnueli, Lichtenstein, 1990
  - Timed Propositional Temporal Logic, Alur and Henzinger, 1991
- CTL-extensions
  - TCTL Timed Computational Tree Logic, Alur and Dill, 1989
- Interval temporal logic
  - Duration Calculus, Chaochen, Hoare and Ravn, 1991

Approaches to continuous time model checking

Timed LTL

- Timed extension of SPIN, based on Büchi automata, 1996

Timed CTL (or a subset of it)

- UPPAAL, K. Larsen et al., networks of timed automata, 1997
- KRONOS, S. Yovine, idem, 1997
- HYTECH, T.A. Henzinger, networks of hybrid automata, 1997
- and many others ....

UPPAAL2K

Integrated collection of tools for:

- symbolic simulation and
- automatic verification of real-time systems

Developed at:

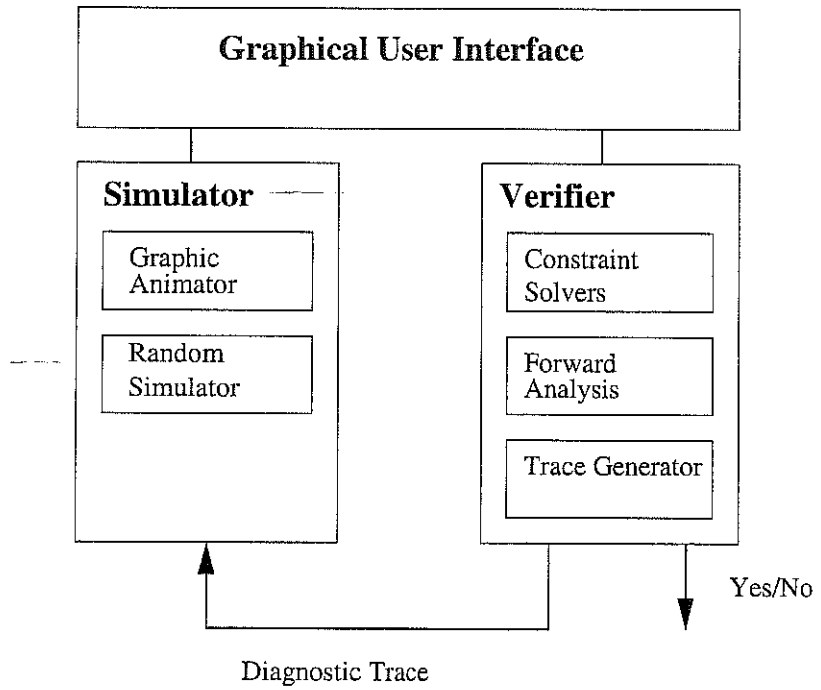
- Dept. of Computing Systems at **Upp**sala University (Sweden)
- BRICS at **Aal**borg (Denmark)

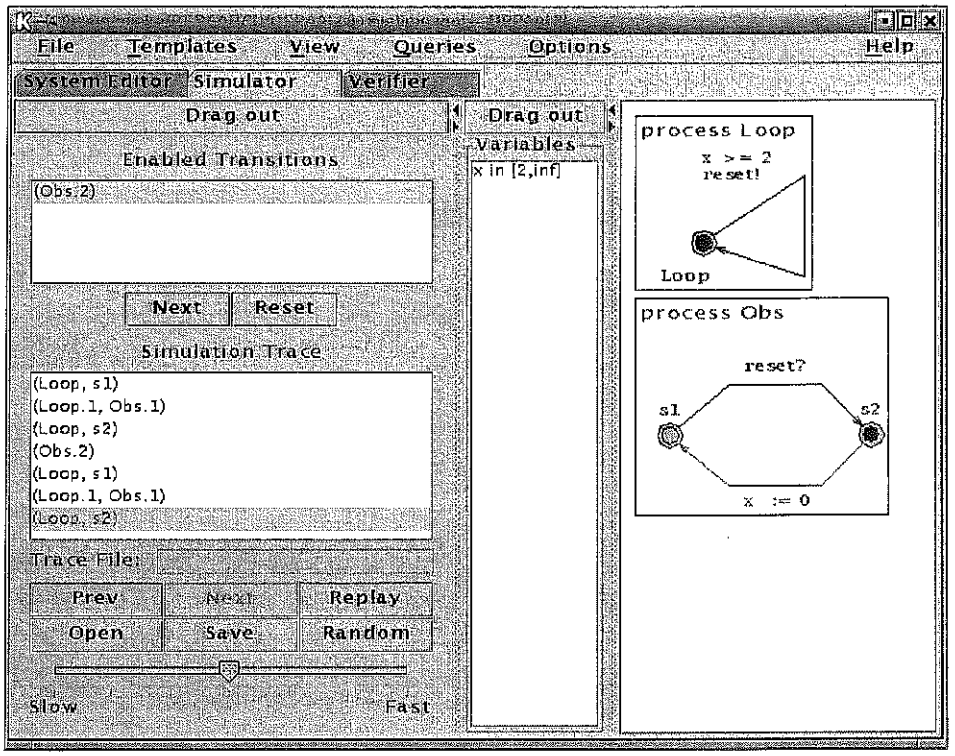
By:

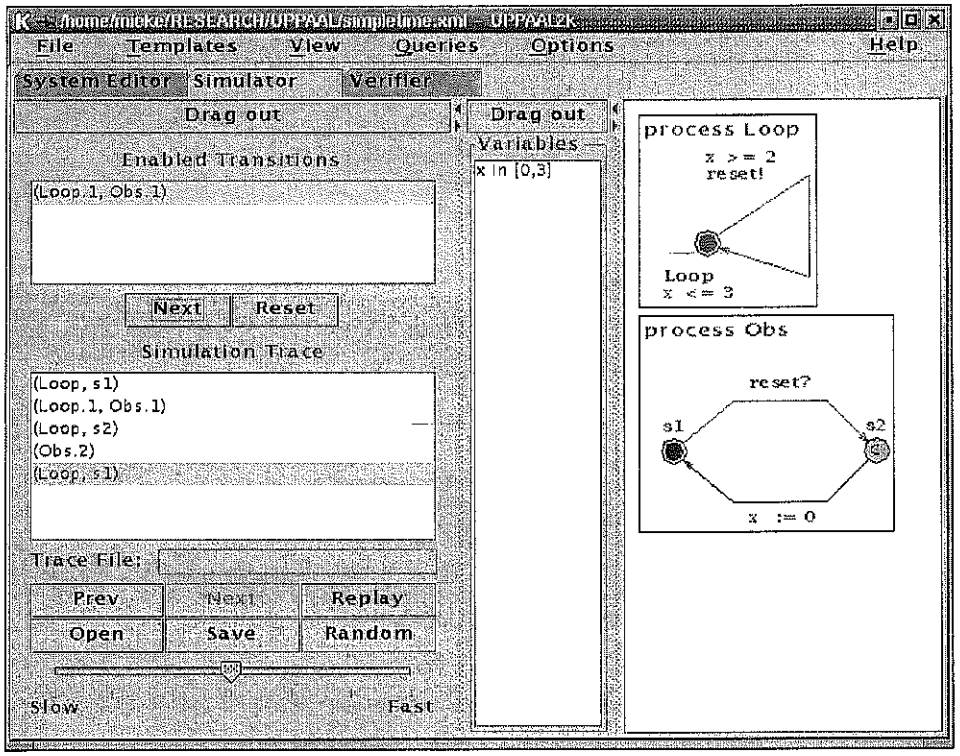
- Prof. Kim G. Larsen, Aalborg
- Prof. Wang Yi, Uppsala
- Dr. Paul Petterson, Uppsala

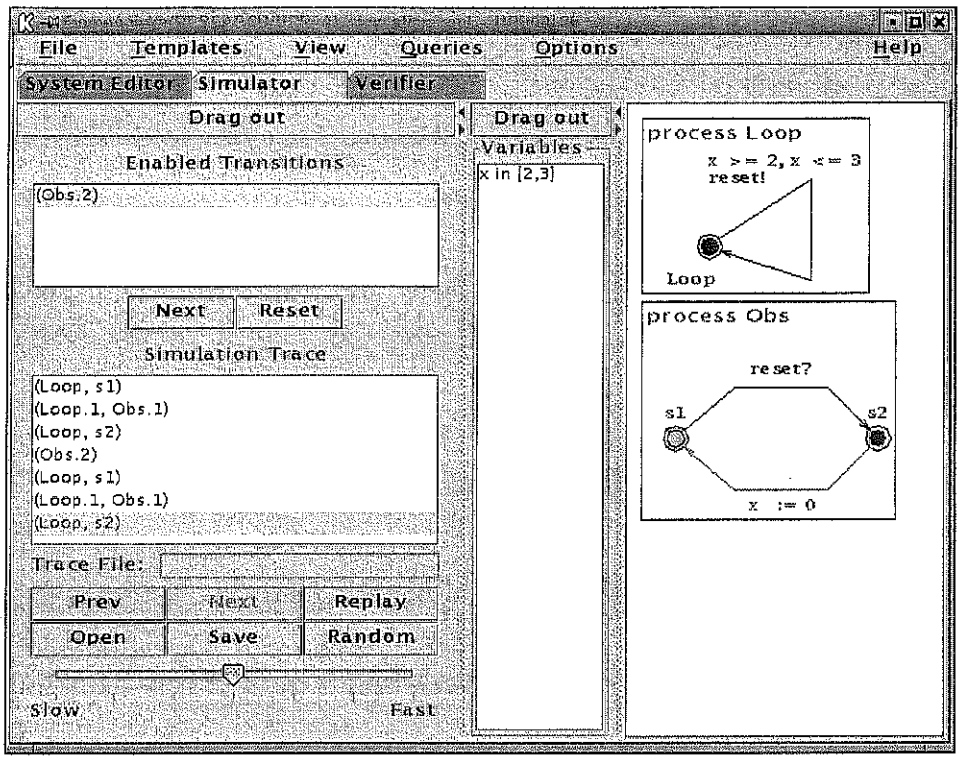
Web page: **http://www.docs.uu.se/rtmv/uppaal**

Overview of UPPAAL tools




File
Templates
View
Queries
Options
Help
System Editor
Simulator
Verifier
Drag out
Enabled Transitions
(Obs.2)
Next
Reset
Simulation Trace
(Loop, s1)
(Loop.1, Obs.1)
(Loop, s2)
(Obs.2)
(Loop, s1)
(Loop.1, Obs.1)
(Loop, s2)
Trace File:
Prev
Next
Replay
Open
Save
Random
Slow
Fast
Drag out
Variables
x in [2,inf]
process Loop
x >= 2
reset!
Loop
process Obs
reset?
s1
s2
x := 0


File
Templates
View
Queries
Options
Help
System Editor
Simulator
Verifier
Drag out
Enabled Transitions
(Loop.1, Obs.1)
Next
Reset
Simulation Trace
(Loop, s1)
(Loop.1, Obs.1)
(Loop, s2)
(Obs.2)
(Loop, s1)
Trace File:
Prev
Next
Replay
Open
Save
Random
Slow
Fast
Drag out
Variables
x in [0,3]
process Loop
x >= 2
reset!
Loop
x <= 3
process Obs
reset?
s1
s2
x := 0



Multipart Synchronisation

Committed states:

- Syntax: location name annotated with **C**
- Semantics:
  - committed locations have to be left **without delay** and
  - they **do not allow interference** with any other transition
- Purpose: mimick atomicity of series of transitions or multipart synchronisation

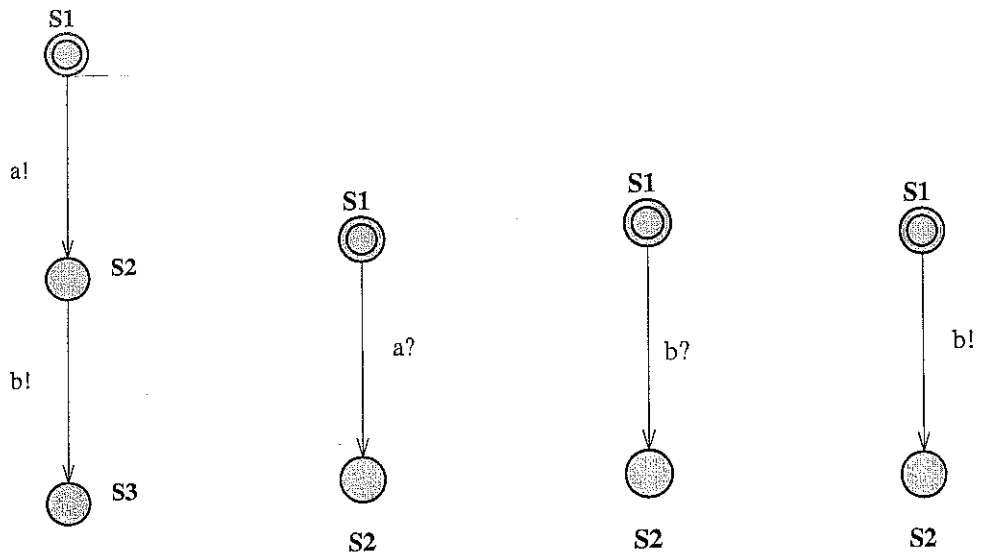
WARNING: at most **one** location **at a time** is allowed to be a committed location!! (Except in case of synchronisation)

Urgent channels

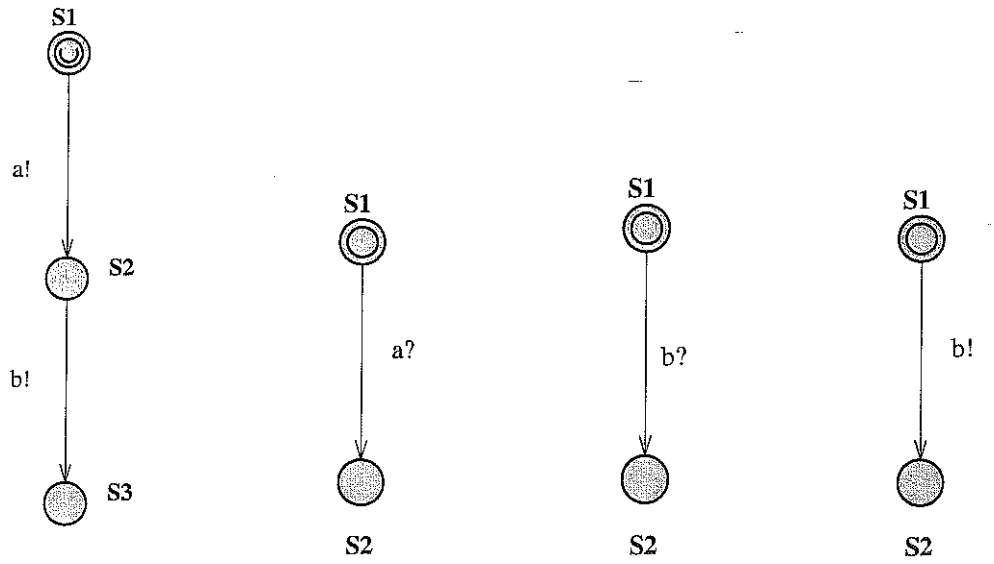
- Syntax: declaration of urgent chan in global declarations section or marking location as urgent
- Semantics: let transitions occur as soon as possible (i.e. without delay), but not necessarily without interleaving with other enabled transitions.

Note: Internal transitions are **NOT** urgent.

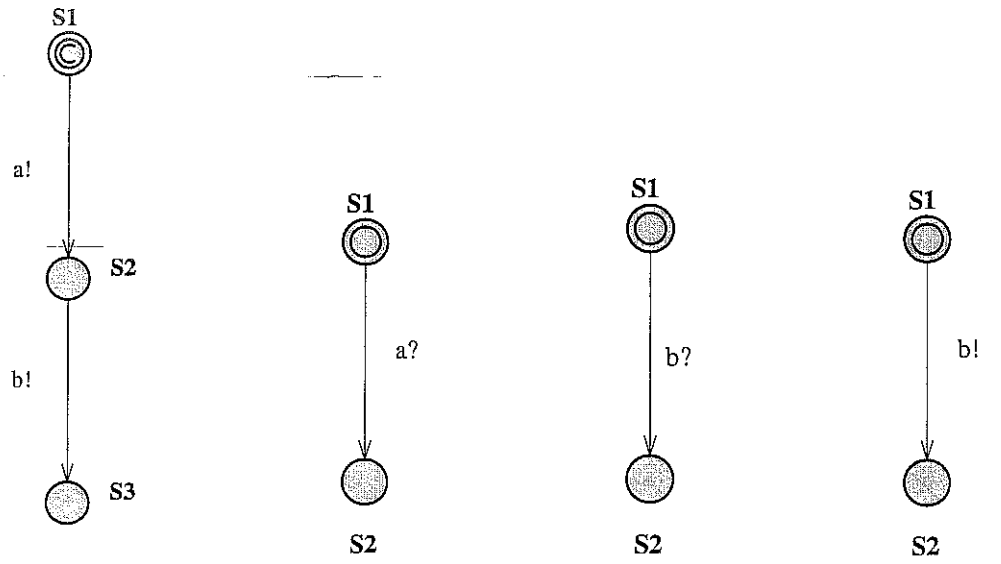
Examples: multipart synchronisation (0)



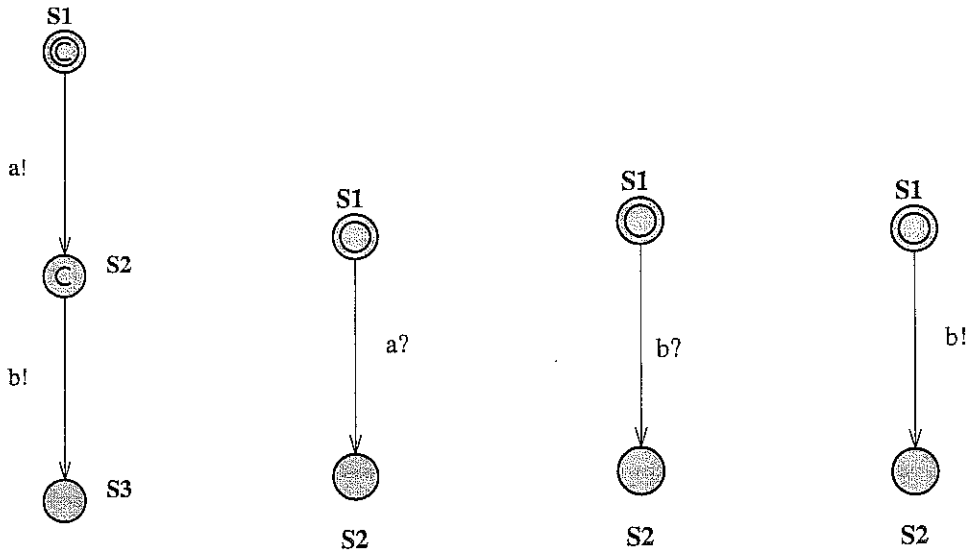
Examples: multipart synchronisation (1)



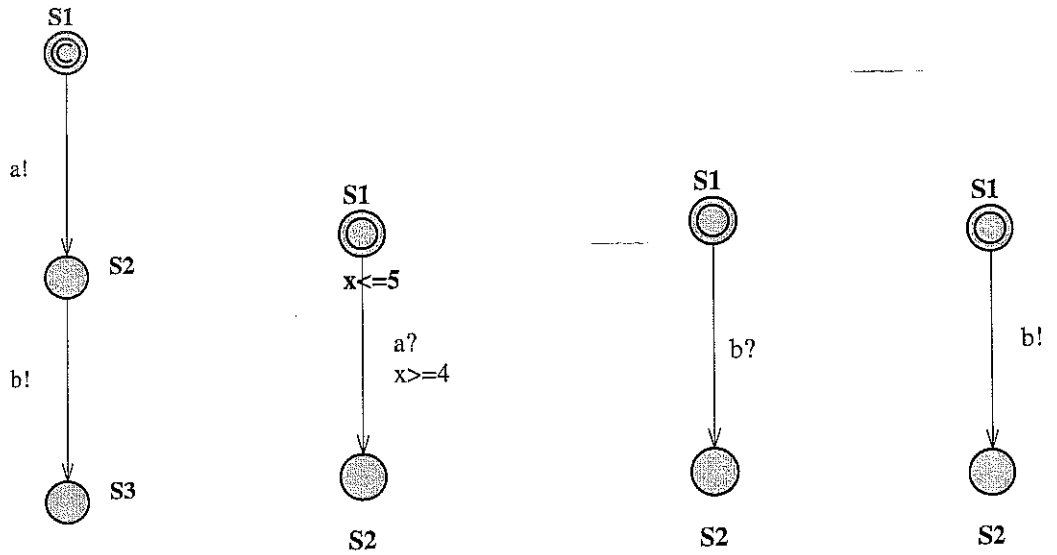
Examples: multipart synchronisation (2)



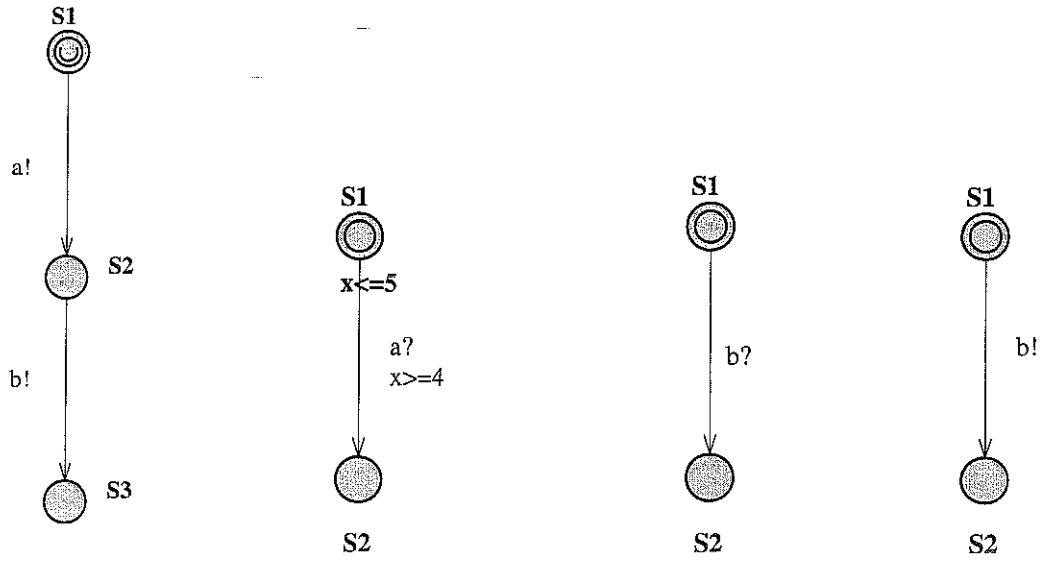
Examples: multipart synchronisation (3)



Examples: committed state timelock (1)



Examples: urgent location (1)



Formal Semantics of Networks of Automata

The semantics of a network of timed automata A is an *infinite* Labelled Transition System $(S, \sigma_0, \mathcal{T})$ where:

- S is the set of states $< \bar{l}, \bar{v} >$ where:
  - $\bar{l}$ control vector of system A
  - $\bar{v}$ a valuation of the clocks and variables of the system
- σ_0 the initial state
- $\mathcal{T}$ a subset of $S \times S$ the transition relation (next slide)

Notation:
l_i current location of automaton A_i
l_i' some other location of automaton A_i
g_i, r_i guard and reset set of $l_i \longrightarrow l_i'$ and α_i the action label
$I(l_i)$ invariant of l_i

Transition Relation

Internal transition of one automaton

- $\langle \bar{l}, v \rangle \rightsquigarrow \langle \bar{l}[l_i'/l_i], r_i(v) \rangle$ if i such that
  1. $l_i \xrightarrow{g_i,\tau,r_i} l_i'$ (is a possible internal transition)
  2. $g_i(v)$ (the transition is enabled)
  3. $Comm(l_k)$ implies $k = i$ (only l_i is committed)

Synchronisation

- $\langle \bar{l}, v \rangle \rightsquigarrow \langle \bar{l}[l_i'/l_i, l_j'/l_j], (r_i \cup r_j)(v) \rangle$ if i,j such that
  1. $l_i \xrightarrow{g_i,\alpha!,r_i} l_i'$ and $l_j \xrightarrow{g_j,\alpha?,r_j} l_j'$ for some α
  2. $g_i(v)$ and $g_j(v)$ (both are enabled)
  3. $Comm(l_k)$ implies that $k = i$ or $k = j$ (only l_i and l_j committed)

Delay

- $\langle \bar{l}, v \rangle \rightsquigarrow \langle \bar{l}, v \oplus d \rangle$
  1. there is no $i,j,\alpha \in U$ such that $l_i \xrightarrow{g_i,\alpha!,r_i} l_i'$ and $l_j \xrightarrow{g_j,\alpha?,r_j} l_j'$ (no urgent synch)
  2. there is no i such that $Comm(l_i)$
  3. for all i: $I(l_i)(v \oplus d)$ (delay is allowed)

UPPAAL: Symbolic Simulation

- Only **observable** and **internal** transitions are shown so:
- **No** delay-transitions are simulated explicitly
- Values of variables and clocks **can** be observed
- All based on the notion of *Region Automata*

Region Automata very informal:

- Uses the fact that clocks can only be reset to natural number values
- Passing of time may create changes in the set of enabled actions
- but (important!!) **only** at certain points in time.
- So, time can be divided into segments (regions) during which no changes occur in the set of enabled transitions in a particular state.
- This abstraction gives rise to an abstract automaton called Region Automaton with a **finite(!)** number of states, amenable to standard model checking techniques.

UPPAAL: Real-time Verification

Properties that can be analysed are of the following forms:

$$\Phi ::= A[]\beta \mid E <> \beta \mid A <> \beta \mid E[]\beta \mid A[]not\ \ deadlock$$
$$\beta ::= a \mid \beta_1 \texttt{ and } \beta_2 \mid \beta_1 \texttt{ or } \beta_2 \mid \beta_1 \texttt{ implies } \beta_2 \mid \beta_1 - - > \beta_2 \mid \texttt{ not } \beta$$

Where a is an atomic formula of the following form:

- $A_i.l$ where A_i a process and l a location of A_i.
- $v_i \sim n$ where v_i is a variable, n a natural number and $\sim$ in $\{<, >, <=, >=, ==\}$

Informal meaning of properties

Safety properties:

- $A[]p$ means: for all paths (execution traces), p will always hold
- $E <> p$ means: there exists a path where p will eventually hold

Liveness properties:

- $A <> p$ means: for all paths, p will eventually hold
- $E[]p$ means: there exists a path where p always holds
- $p - - > q$ means: whenever p holds, q will eventually hold

$A[]not\ \ deadlock$ checks for absence of deadlock

Satisfaction relation

The satisfaction relation is given by rules like:

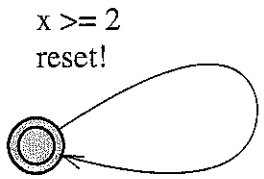
- $< \bar{l}, v > \models E <> \beta \Leftrightarrow \exists < \bar{l}', v' > . < \bar{l}, v > \rightsquigarrow^* < \bar{l}', v' > \wedge < \bar{l}', v' > \models \beta$
- $< \bar{l}, v > \models A[]\beta \Leftrightarrow \forall < \bar{l}', v' > . < \bar{l}, v > \rightsquigarrow^* < \bar{l}', v' > \Rightarrow < \bar{l}', v' > \models \beta$
- $< \bar{l}, v > \models c \Leftrightarrow c(v)$
- $< \bar{l}, v > \models A_i.l \Leftrightarrow l_i = l$

Exercise: formulate the rules for $E[]\beta$ and $A <> \beta$

Following are some verification examples

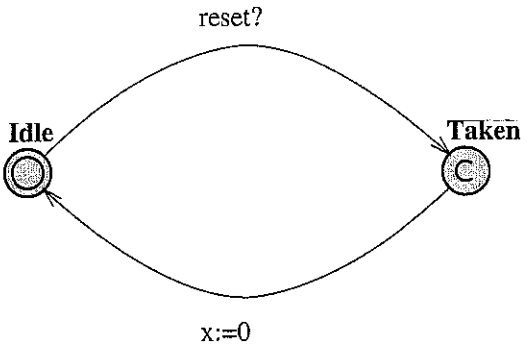
Examples (0)

Process Looping



Loop

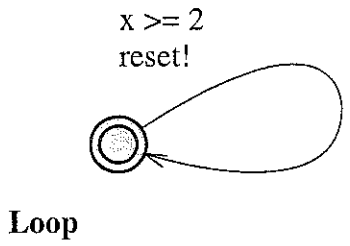
Process Obs



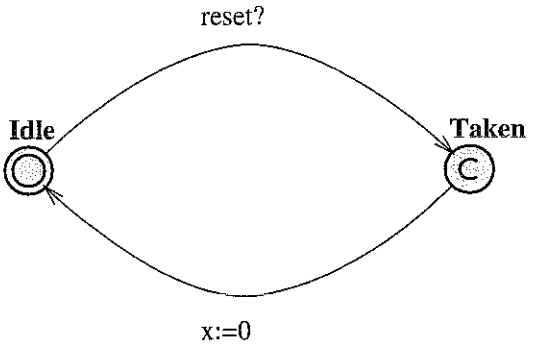

```
A[] Obs.taken imply x>= 2
```

Examples (1)

Process Looping



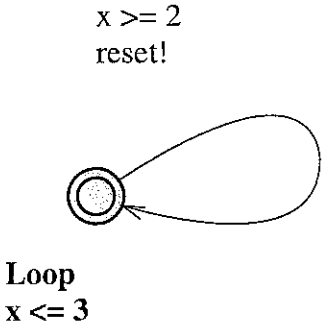
Process Obs



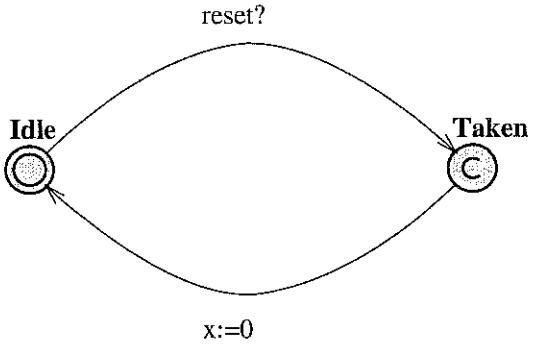

```
E<> Obs.idle and X>3
```


Examples (2)

Process Looping



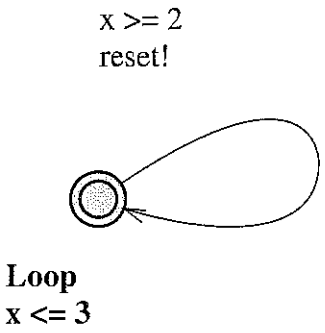
Process Obs



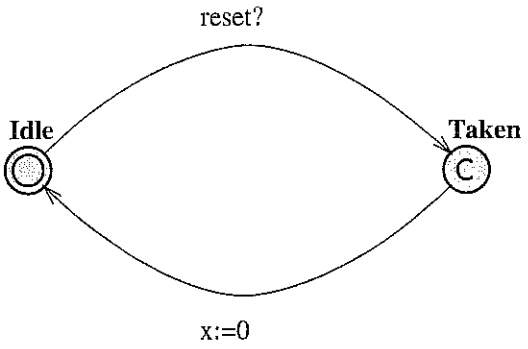

```
A[] Obs.taken imply (x>=2 and x<=3)
```

Examples (3)

Process Looping



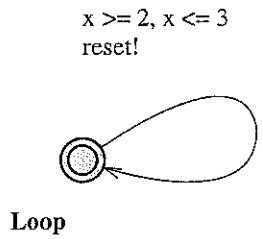
Process Obs



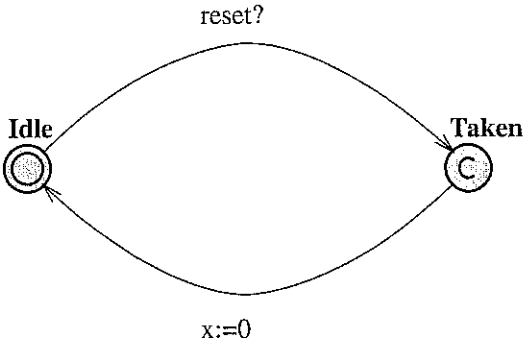
`E<> Obs.idle and x>2` and `A[] Obs.idle imply x<=3`

Examples (4)

Process Looping

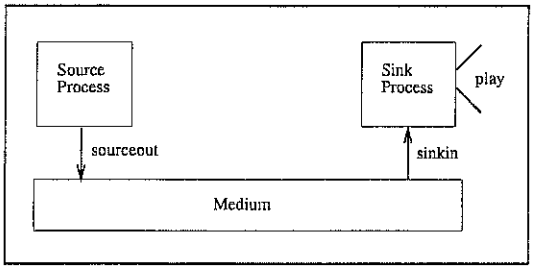


Process Obs



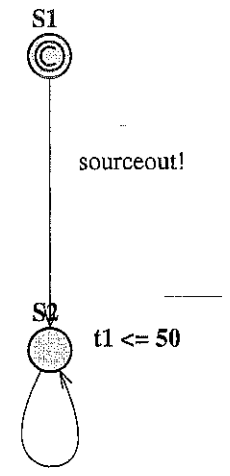
`E<> Obs.idle and x>2` holds, but `A[] Obs.idle imply x<=3` is FALSE!

Case study: Quality of Service of a Media Stream

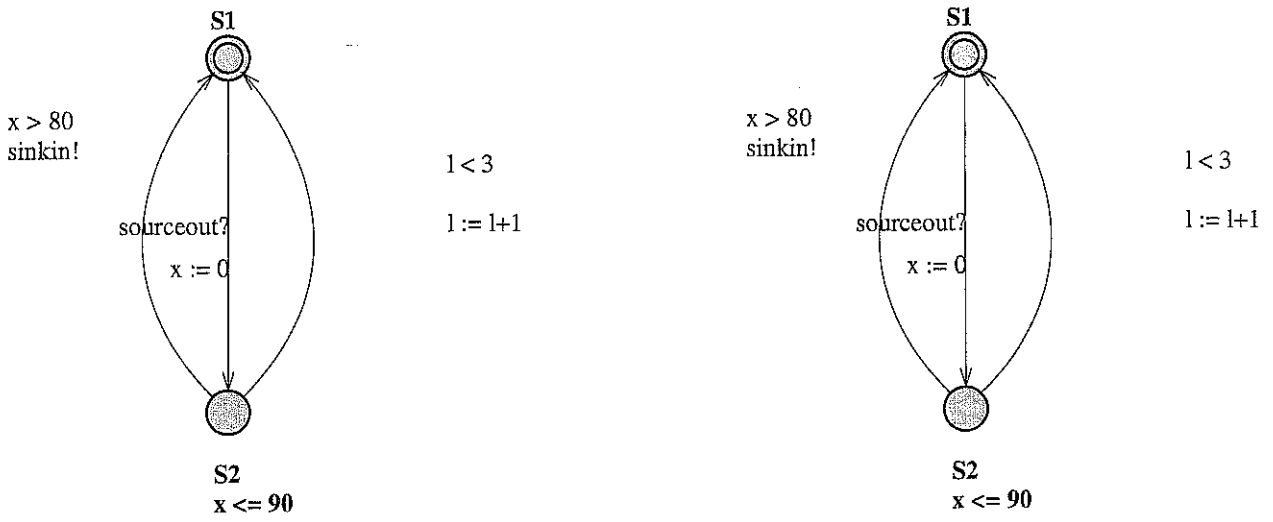


- Source emits message every 50 ms, so 20 messages per second
- Channel latency is between 80 and 90 ms
- Channel may loose messages, but less than 20%
- A message is lost if it does not arrive within 90 ms
- Sink receives messages, but every message takes 5 ms to be processed
- An error should be generated if less than 15 messages per second received

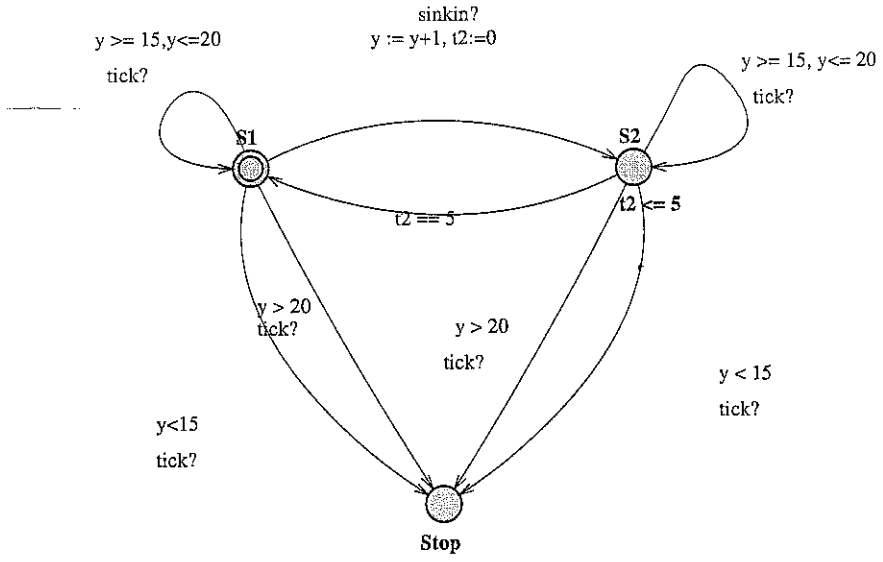
Modelling the behaviour of the Source



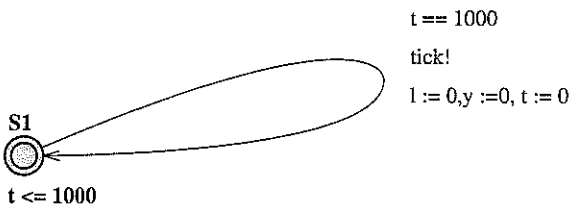
Modelling the behaviour of the Channel



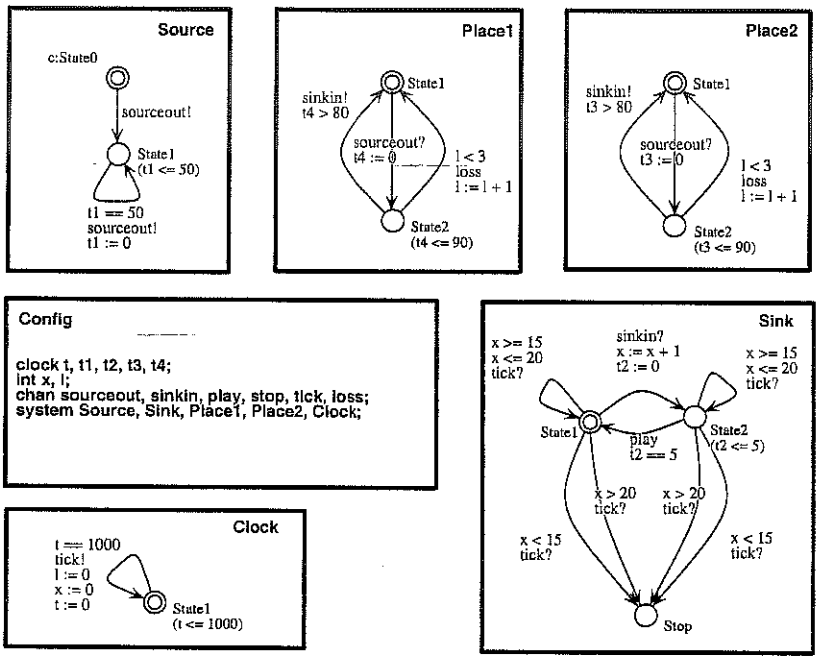
Modelling the behaviour of the Sink



Modelling the throughput Monitor



Full specification (Uppaal 1.4)



Simulation and Verification

1. Emitted messages should always be accepted by the channel
2. The throughput should remain between 15 and 20 frames per second
3. What is the maximal loss given the required throughput?
4. The system should never deadlock
5. The timer of the monitor should never go beyond 1000 ms
6. Anything else?

Simulation and Verification

1. Emittes messages should always be accepted by the channel
2. The throughput should remain between 15 and 20 frames per second
3. What is the maximal loss given the required throughput?
4. The system should never deadlock
5. The timer of the monitor should never go beyond 1000 ms
6. Anybody else?

1. `E<>(Place1.S2 and Place2.S2 and t1==50)` must be FALSE
2. `E<>(Sink.stop)` must be FALSE
3. Combination of (1) and changing values for `l`
4. `A[](not deadlock)` should be TRUE
5. `A[](t1 <= 1000)` should be TRUE

Uppaal2k Verification Options

Menus and documentation of UPPAAL2K in **HELP**

Options menu:

- Search Order
- State Space Reduction
- State Space Representation
- Clock Reduction
- Re-use State Space
- Diagnostic Trace

Some options may interfere and are automatically switched off when others are selected.

Search Order

Performs (symbolic) state space exploration:

- breadth first: could help to find shorter path to problem
- depth first

State Space Reduction

Defines preferences in dealing with Space-Time trade-off. In some specifications control structure analysis may help to reduce space needed for verification.

- none: no analysis to reduce space
- conservative:
- aggressive: maximal space reduction, may need more time to verify

State Space Representation

Different options:

- DBM (Difference Bound Matrices)
- Compact Data Structure
- Under Approximation (bit state hashing)
- Over Approximation (convex hull)

Clock reduction

This technique tries to re-use clocks defined in the specification, but that are no longer used after a certain time.

The number of clocks used in a specification plays a major role in the complexity of the model-checking algorithm. In fact, the amount of space needed is n^2 where n number of clocks.

Re-use State Space

Used when more than one property of a system is checked at once.

Model-checking tries to re-use the state-space generated for one property when verifying another property.

Diagnostic Trace

When set, generates when possible, a trace from the starting state to the state where a property is (not) violated. The trace is automatically loaded into the simulator for further analysis of the problem.

From Reachability to Bounded Liveness

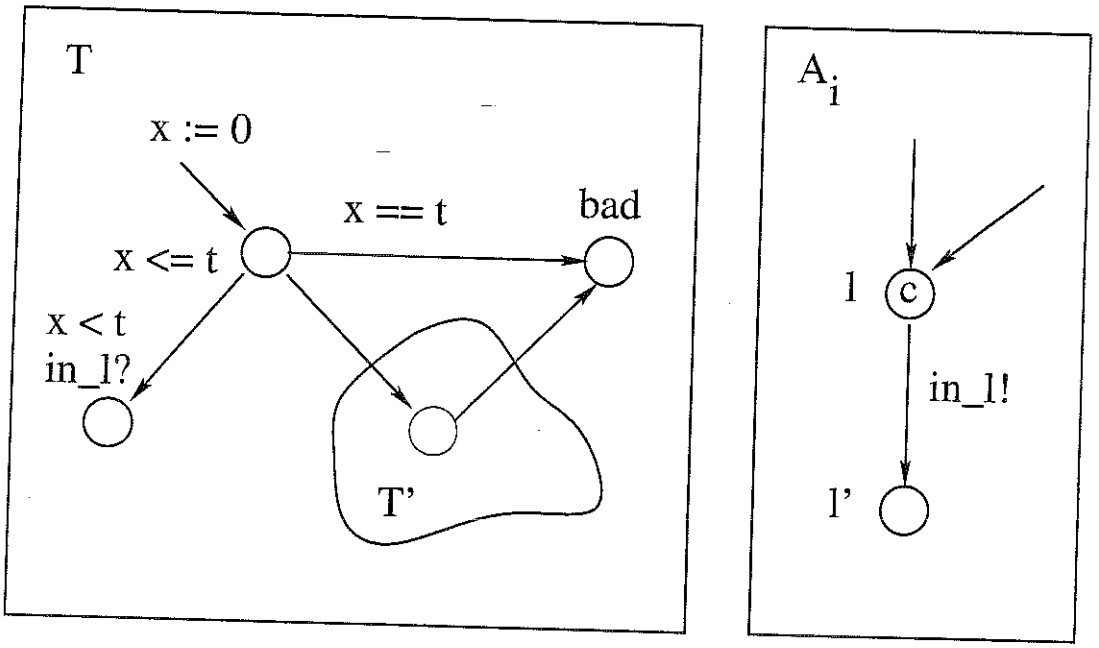
Example:

"ϕ holds until property a becomes true before time t"

Property a may be for example: Automaton A_i at location l.
Above property cannot be expressed directly in the logic.

Solution: use of **Test-Automata**

Test-Automata



Checking liveness using reachability

(S satisfies $\Psi = \phi$**Until**$_{<t}\ a$) iff ($S \mid T$ satisfies $A[]not(T.bad)$)

Conjecture: all bounded liveness properties can be translated into reachability properties.

Further Reading

- Model-checking techniques:
  - Clark, Grumberg, Peled; Model Checking; MIT Press, 1999
  - McMillan; SMV Model Checking; Kluwer AP, 1993
  - J. Katoen, Concepts, Algorithms, and Tools for MC, 1999
  - Journal on STTT (Software Tools for Technology Transfer), 1, Springer, 1997
- UPPAAL:
  - P. Petterson, PhD thesis (electr. available)
  - UPPAAL2K Web-page