Model checking and Process algebras

Cinzia Bernardeschi

Department of Information Engineering

University of Pisa

FMSS, 2019-2020

Model checking

Mechanical checking of the satisfaction of a logic formula on the model of the behaviour of the system

Model construction:

- Kripke structure (Transition System - TS)
  $S = \{s_1, \cdots s_n\}$
  $AP = \{p_1, p_2, ...p_k\}$ set of atomic propositions
  $L : S \rightarrow Powerset(AP)\}$

- Labelled transition system (LTS)
  $S = \{s_1, \cdots s_n\}$
  $A = \{a_1, ..., a_m\}$ seo of actions
  $\rightarrow$: $S \times A \times S$

LTS can be generated by process algebras specifications.

Process algebras

Process algebras are a standard tool for the modelling of concurrent systems.

Assume models given using the Calculus of Communicating Processes (CCS) [Milner, 89].

- A system consists of a set of communicating processes;
- each process executes actions, and synchronizes with other processes.
- Moreover, a special action τ denotes an unobservable action and model internal process actions or internal communications.

Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)

CCS

Syntax and informal semantics of CCS operators.

stop	Inactive process	A process which does nothing
$a : P$	Action prefix	Action a is performed and then process P is executed
$P + Q$	Nondeterministic choice	Alternative choice between the behaviour of process P and that of Q
$P \mid\mid Q$	Parallel Composition	Interleaved execution of process P and process Q
$P \setminus a$	Action restriction	Behaves like P apart from action a that can only be performed within a communication
$P[a/b]$	Action renaming	Behaves like P apart from action a that is renamed b

CCS

- The specification is based on a set *Act* of elementary actions that processes can perform and on a set of operators that permit to build complex processes from simpler ones.
- The special action τ, not belonging to *Act*, represents the unobservable action and is used to model internal process actions or to hide actions to the external environment.
- We denote by $Obs(P)$ the set of observable actions of the process P.

CCS

An inactive process is specified by the **stop** operator.

The action prefix operator specifies the execution of actions in sequence.

The nondeterministic choice operator indicates that a process can choose between the behaviour of several processes.

Parallel composition of two processes corresponds to the interleaved execution of the two processes.

The restriction operator is used to specify processes which synchronise on actions (communication).

A communication transforms the couple of actions executed together into the internal action τ.

The relabeling operator transforms an action into another action.

Labelled Transition Systems

The semantics of process algebras are Labeled Transition Systems (LTSs) which describe the behavior of a process in terms of states, and labeled transitions, which relate states.
An LTS describes sequential nondeterministic behaviours. More formally,

Definition

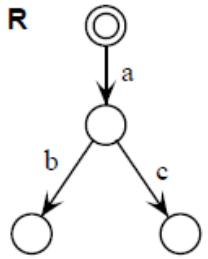
An LTS is a 4-tuple $\mathcal{A} = (X, x^0, Act \cup \{\tau\}, \rightarrow)$, where: X is a finite set of states; x^0 is the initial state; Act is a finite set of observable actions; $\rightarrow \subseteq X \times Act \cup \{\tau\} \times X$ is the transition relation.

We denote by $x \xrightarrow{a} x'$, $a \in Act \cup \{\tau\}$, the transition from the state x to the state x' by executing action a.

Labelled Transition system

Let us consider the process R below. Process R executes action *a* and then may execute action *b* or action *c*, and then stops.

R = a: (R1 + R2)
R1 = b: stop
R2= c: stop

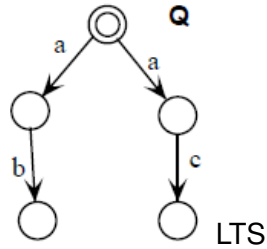


LTS

Labelled Transition System

Let us consider process Q below. Q, differently from R, executes the choice between action *b* and *c* hen performing action *a*.

Q = Q1 + Q2
Q1 = a: b: stop
Q2= a: c: stop

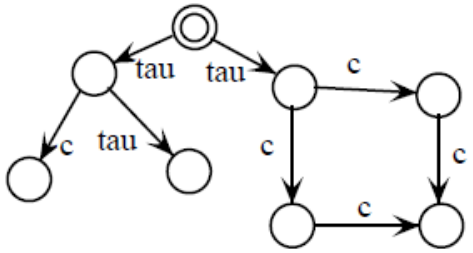


LTS

Labelled Transition system

Let us consider the system described by the following process:
$P = (R \parallel Q) \setminus a \setminus b$

LTS of P



Every state of the LTS represents the combined current states of the subsystems components.
In the initial state, only *a* can be executed. Since *a* is a synchronisation action, *tau* is shown in the LTS.
Then following the left (right) edge of the LTS of *Q*, the behaviour of P is described by the left (right) subtree.

Weak bisimulation equivalence

Definition

Given an LTS $\mathcal{A} = (X, x^0, Act \cup \{\tau\}, \rightarrow)$, a weak bisimulation equivalence over X is a maximal binary symmetric relation $\mathcal{S}$ such that, for any $x, y \in X$, we have $x \mathcal{S} y$ if and only if: $\forall a \in Act \cup \{\tau\}$,

1. $x \stackrel{a}{\Rightarrow} x' \rightarrow (\exists y', y \stackrel{a}{\Rightarrow} y' \wedge x' \mathcal{S} y')$
2. $y \stackrel{a}{\Rightarrow} y' \rightarrow (\exists x', x \stackrel{a}{\Rightarrow} x' \wedge x' \mathcal{S} y')$

LTS describe the behavior of the processes in details, including their internal computations.

Weak bisimulation equivalence

A widely used equivalence is *weak bisimulation*, or *observational* equivalence, first introduced by Milner, based on the idea that only the externally observable actions of a system are relevant in its interaction with the environment

To abstract unobservable moves during observation, the weak transition relation $\stackrel{a}{\Rightarrow}$ is used.

We have: $\forall a \in Act, \stackrel{a}{\Rightarrow} = (\stackrel{\tau}{\rightarrow})^\star \stackrel{a}{\rightarrow} (\stackrel{\tau}{\rightarrow})^\star$, where $\star$ means zero or any number of times.

Two states x and y are considered as observational equivalent if and only if x and y must be able to perform equal sequences of actions evolving to equal (up to $\mathcal{S}$) states.

Weak bisimulation equivalence

The relation between states of a transition system can be easily extended to a relation between two distinct transition systems.

Two systems are then observationally equivalent whenever no observation can distinguish them.

Definition

Given two processes R and Q, they are called *observational equivalent* if and only if a weak bisimulation $\mathcal{S}$ exists which relates the initial states of the LTSs which describe their behavior and we write $R \approx Q$.

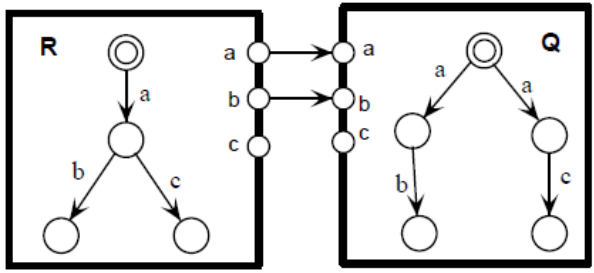
Observational equivalence ($\approx$ in the following) is then defined upon the $\stackrel{a}{\Rightarrow}$ relation.

Weak bisimulation equivalence

Are processes R and Q observational equivalent ($R \approx Q$) ?

Graphical notation

The graphical specification of process P is the following.



Weak bisimulation equivalence

Processes R and Q are not observational equivalent, since there exists no state in Q bisimilar to the state in R reached after having executed action a (in this state both action b and c can be performed).

Expressing properties

The temporal logic ACTL (Action-based Computation Tree Logic).

ACTL is an action-based version of the branching time temporal logic CTL.

ACTL has the advantage that, since it is based on actions rather than states, it is naturally interpreted over LTSs.

ACTL

The formulae of ACTL are *action formulae*, *state formulae* and *path formulae*.

An *action formula* permits expressing constraints on the actions that can be observed.

A *state formula* gives a characterization about the possible ways an execution can proceed after a state has been reached.

A *path formula* states properties of an execution.

The truth or falsity of a formula refers to a satisfiability relation over LTSs, denoted $\models$.

Syntax and informal semantics of the used ACTL operators

Action formulae

$\chi ::= \mathit{true}$	any observable action
a	the observable action a
$\sim \chi$	any observable action different from χ
$\chi \mid \chi'$	either χ or χ'

State formulae

$\phi ::= \mathit{true}$	any behaviour is possible
$\sim \phi$	ϕ is impossible
$\phi \,\&\, \phi'$	ϕ and ϕ'
$E\gamma$	there exists an execution in which γ
$A\gamma$	for every execution γ
$< a > \phi$	there exists a next state reachable with a, in which ϕ
$[a]\phi$	for all next states reachable with a, ϕ holds

Path formulae

$\gamma ::= G\phi$	at any time ϕ
$F\phi$	there is a time in which ϕ
$[\phi\{\chi\}U\{\chi'\}\phi']$	at any time χ is performed and also ϕ, *until* χ' is performed and then ϕ'

ACTL formulae

In the table, a is an action belonging to the set Act of actions executable by the system, $\sim$ is the *negation* operator, E and A are the existential and universal path quantifiers, while U is the *until* operators.

For example, the formula:
$AG([a](< b > true \ \& < c > true))$
states that the system, after having executed the action a, has always the possibility of performing both b and c. This formula is true on the LTS of process R and it is false on the LTS of process Q.

The model checker tool provides a counter-example facility. In the case of satisfied formulae this facility reports a path which verifies the formula; otherwise a path which does not verify the formula is given.

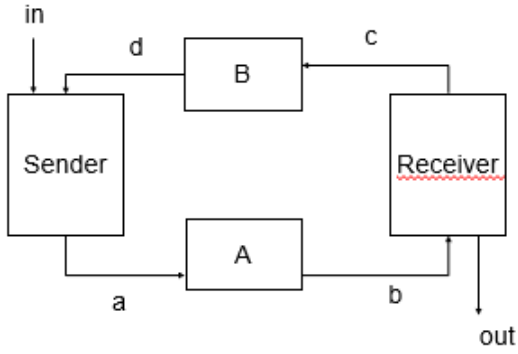
Alternating-bit protocol

The purpose of the protocol is ensuring reliable communication over a medium which may loose messages. A possible implementation of the protocol consists of four processes: the Sender, the Receiver, and two communication channels: one for the delivery of the message, and another for the acknowledgment of message reception.

Sender and Receiver use the value of one bit to identify a message, so that the identifier bit of each message is the complement of the preceding message's bit; a new message is not sent until the sender receives acknowledgment of the current message.

Since the channels can loose messages, both the Sender and the Receiver resend the same message or, respectively, acknowledgment repeatedly until the acknowledgment is received.

Protocol schema



$a : a_0, a_1$
$b : b_0, b_1$
$c : c_0, c_1$
$d : d_0, d_1$

Actions

Upon an *in* action at the system's external interface, the *Sender* sends the message to the *Receiver* through channel *A*.

Synchronization on action $a0$ or $a1$ depending on the current value of the alternating bit (the first message is identified as 0).

Upon receiving the message, the *Receiver* executes *out*, meaning that the message is available at the interface.

Next, the *Receiver* sends the acknowledgment by synchronizing with channel B on action $c0$ or $c1$ according to the value of the identifier bit of the received message.

Alternating-bit protocol

$$P = in.out.P$$

The system is specified by the process *Sys*.

$$Sys = (S_0|A|B|R_1)\backslash L$$

- S_0 is the Sender whose alternating bit is 0;
- R_1 is the Receiver, whose alternating bit is 1;
- A is the delivery channel
- B is the ack channel
- $L = \{a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1\}$

Questions

Some questions:

Are P and Sys weak bisimulation equivalent ?

$$P \approx (S_0|A|B|R_1)/L$$

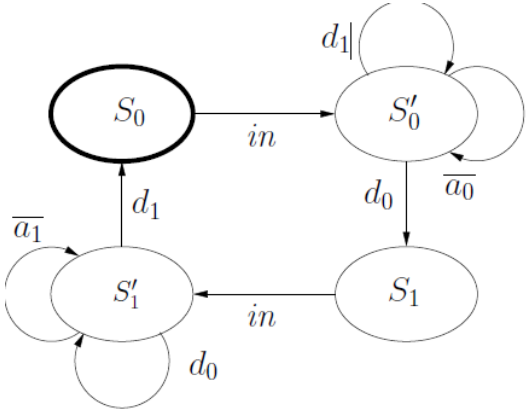
Is it always possible to execute out?

Omission of messages or acknowledgments is represented by the τ actions in the processes for the channels, which can take a channel from a state to another without executing the corresponding synchronization action.

For clarity, we use the notation: input action a and output action $\bar{a}$

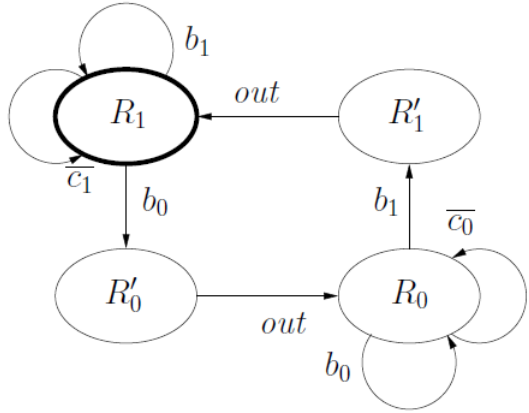
Sender

$$S_0 = in.S'_0$$
$$S'_0 = \overline{a_0}.S'_0 + d_1.S'_0 + d_0.S_1$$
$$S_1 = in.S'_1$$
$$S'_1 = \overline{a_1}.S'_1 + d_0.S'_1 + d_1.S_0$$



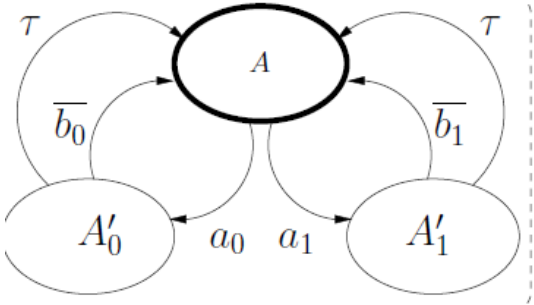
Receiver

$$R_1 = b_0.R'_0 + b_1.R_1 + \overline{c_1}.R_1$$
$$R'_0 = \overline{out}.R_0$$
$$R_0 = b_1.R'_1 + b_0.R_0 + \overline{c_0}.R_0$$
$$R'_1 = \overline{out}.R_1$$



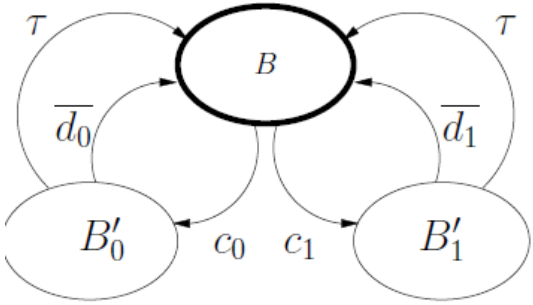
Delivery channel

$$A = a_0.A'_0 + a_1.A'_1$$
$$A'_0 = \overline{b_0}.A + \tau.A$$
$$A'_1 = \overline{b_1}.A + \tau.A$$

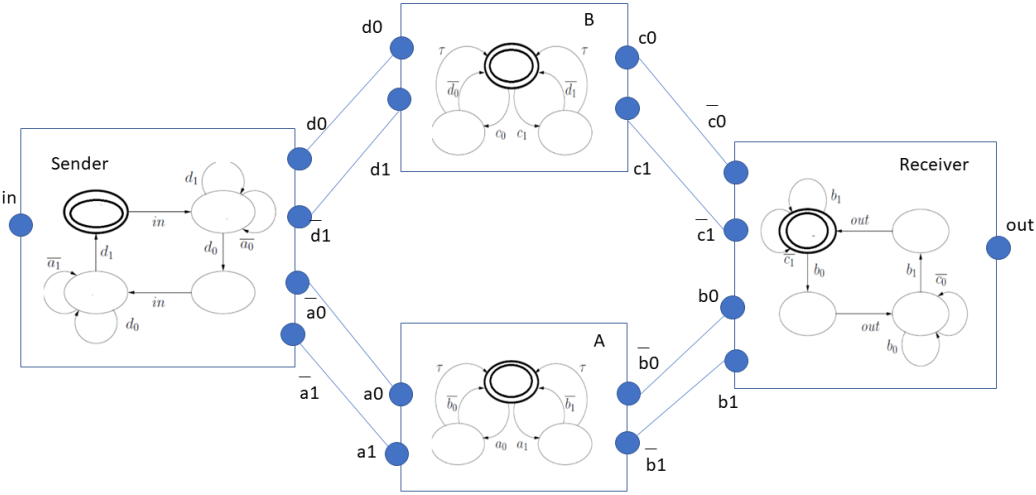


Ack channel

$$B = c_0.B'_0 + c_1.B'_1$$
$$B'_0 = \overline{d_0}.B + \tau.B$$
$$B'_1 = \overline{d_1}.B + \tau.B$$



The system Sys



LTS of the process Sys is generated automatically by the network of processes.

Properties

$$P = in.out.P$$
$$Sys = (S_0|A|B|R_1)\backslash L$$

- P is observational equivalent to Sys: $P \approx Sys$

- It is always possible to execute out: FALSE
a sent by the Sender
a lost by the medium
a re-sent by the Sender
a lost by the medium
.....

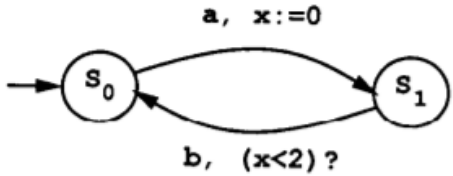
Adding time to models

Timed Automata (Alur and Dill 1994)

- A Timed automaton consists of an automaton A (states and transitions) + clocks
- Clocks progress synchronously with time (time domain R^+)
- a transition consists of a guard, an action and a reset of clocks

Rajeev Alur and David L. Dill, A theory of timed automata, Theoretical Computer Science, 126, 2, 1994

A simple example

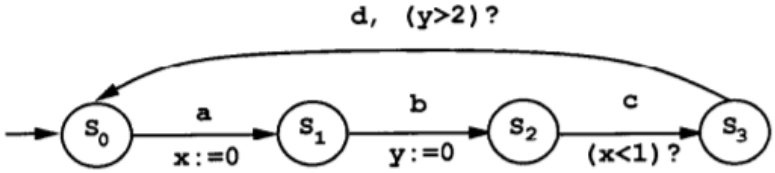


The TA moves from s0 to s1 by reading a. The clock is set to 0 along this transition.
While in state s1, the clock x shows the time lapsed since the occurrence of the last a.
The transition from s1 to s0 is enabled only if this value is less than 2.
The the automaton moves back to state s0 and the cycle is repeated.

Another simple example

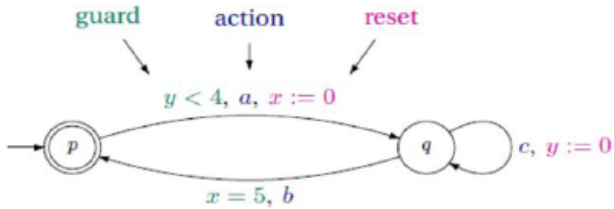
The timed automaton has two clocks: x and y.
The automaton cycles among the states s0, s1, s2, s3

Clock x is set to 0 each time it moves from s0 to s1 reading a.
c happens within time 1 from the preceeding a
clock y is set while reading b
the delay between b and the following d is greater than 2



Timed Automata

x,y: clock



	p	$\xrightarrow[3.2]{a}$	q	$\xrightarrow[5.1]{c}$	q	$\xrightarrow[8.2]{b}$	p	...
value of x	0		0		1.9		5	...
value of y	0		3.2		0		3.1	...

→ timed word $(a, 3.2)(c, 5.1)(b, 8.2)...$

Timed Automata

A transition is labelled with one action, a constraint (guard) and the reset of zero or more clocks (a, g, r). A reset is a clock to be set to zero.

The *state* of a timed automaton is given by the location and the values of the clocks at a given time.
(a, 3.2) (c, 5.1) (b, 8.2) ....
(a, 3.8) (c, ...) (b, ...)

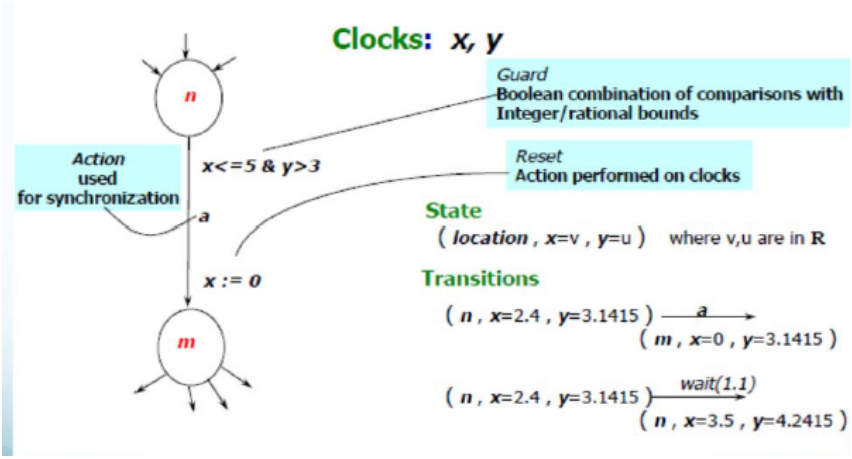
A timed automaton models a system operating in a number of distinct *modes*. Each mode is represented by a location. Mode changes occur as a consequence of the execution of actions.

While the system remains in a given location, progress of time is reflected by the values of the clocks, whose values increase all at the same rate.

Adding invariants

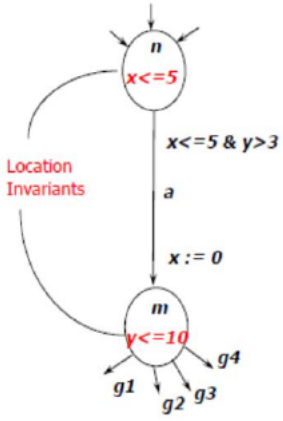
What happens when a does not occur?

Until a does not occur, time alone flows



Adding invariants

Invariants ensure progress



Clocks: *x, y*

Transitions

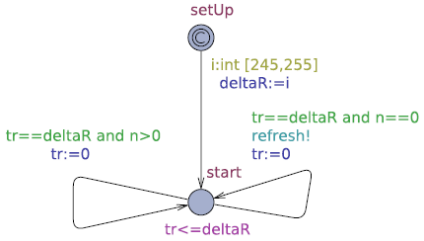
(*n* , *x*=2.4 , *y*=3.1415) ~~wait(3.2)~~ → wait(1.1) → (*n* , *x*=3.5 , *y*=4.2415)

Software tools

- The properties to be proved are expressed in temporal logic. A model checker tool is used to automatically derive proofs.
- The theory of timed automata are implemented in the UPPAAL model checker
- the properties are expressed in Timed Temporal Logic (TCTL)

Example: a TA in UPPAAL

The TA operates on one clock `tr` and a variable `n`. When the clock is equal to the constant `deltaR`, an out edge is executed (`tr <= deltaR` is the state invariant). If variable `n` equals to 0, the TA sends a `refresh` message and resets the clock. Otherwise (`n > 0`), only resets the clock.



The TA models the automatic refresh actions performed by a GUI every `deltaR` time units, in absence of user actions (n==0). Initially, a random value in the interval $[245, 255]$ms is assigned to `deltaR` (4 Hertz)

Networks of Timed Automata

A set of timed automata can be composed to create *Networks of Timed Automata*.

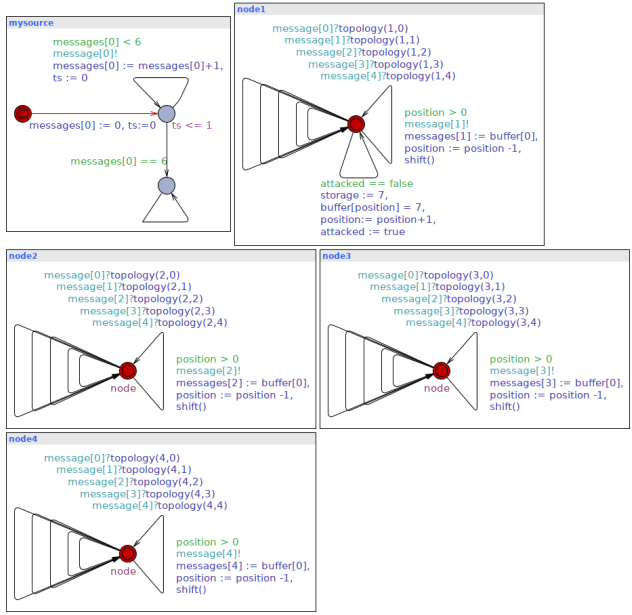
Synchronisation between two automata is modelled by the existence in the network of two edges, one for each of the two automaton, labelled with *complementary* actions, named *input* and *output* actions.

One timed automaton executing an output action synchronises with one or more timed automata, each executing the complementary action. Two edges labelled with complementary actions can be taken only if the constraints on each of them, if any, are satisfied.

Actions not participating in synchronisations, i.e., *internal* actions, are all equivalent with respect to the network behaviour, and are represented by the τ action. Graphically, an input action is denoted by a question mark (*?*), an output action by an exclamation mark (*!*).

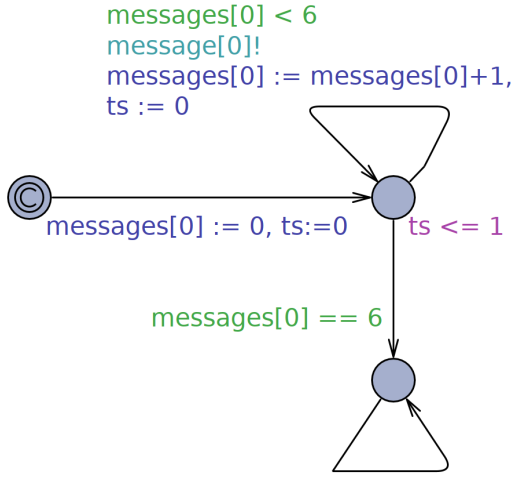
Example

Analysis of security attacks in Wireless Sensor Networks



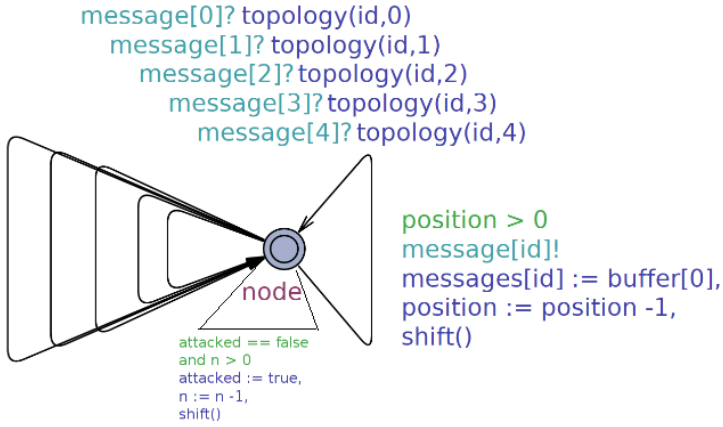
Example

Sender node

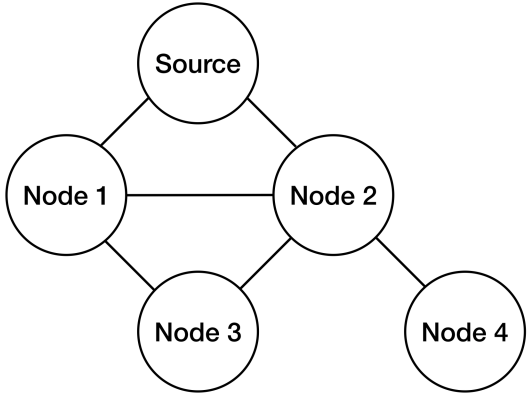


Example

Relay node and drop attack



Network topology



Protocol analysis

Property: Every message is received by all nodes.

The property has been proved to be true in the attack free scenario.

Drop attack. UPPAAL show the following results:

- when the adversary compromises Node 1, the Property is still satisfied (Node 3 receives a copy of the dropped packet from Node 2, thanks to link redundancy);

- when the adversary compromises Node 2, the Property P is not satisfied anymore (Node4 does not receive a copy of the dropped packet since there is no redundancy on links connecting Node 4 with other network nodes)

UPPAAL tests the formulas above against all the possible execution paths o the protocol.