Modelling and Verification of Real-Time Systems

A case study with UPPAAL

Juan Pablo Gruer

Universidad Nacional de Tucumán

Plan

1. Modelling reactive systems
2. The State-Transition Paradigm
3. The UPPAAL modelling Language
4. informal semantics of UPPAAL
5. Verification of reactive systems
6. Expressing properties of UPPAAL models

Sommaire

What and why ?

What is a model ?

A model is an **abstract** representation of a system. A model can be **informal** or **formal**.

- **Abstraction** : all details that are not relevant respectively to the intended usage of the model are not represented in it.
- **Informal model** : is built using informal languages (text, sketches, ...).
- **formal model** : is built using a language with a well-founded semantics, based on logic, mathematics, ...

What and why ?

Why to build models of reactive systems ?

Benefits of building a model during de design process of a reactive system :

- **Communication** : all contributors to the design process can share information about the system (informal or formal).
- **Simulation** : Put the model to "work" in order to grasp or estimate partially how the real system will operate (formal).
- **Verification** : To establish with the strength of a mathematical proof that the model satisfies some property (formal).

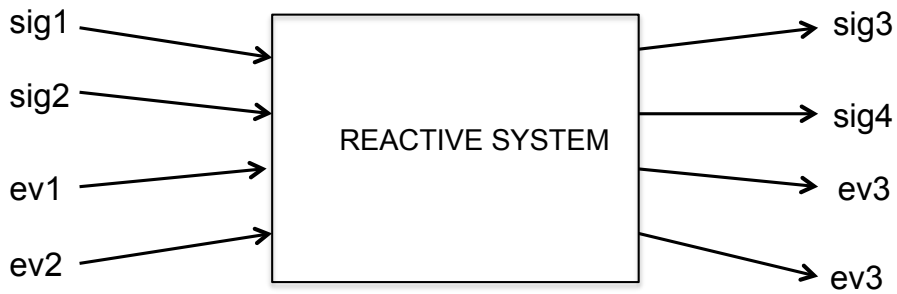
Points of view

A system has different aspects. Each one can be represented by specific modelling languages :

- **Structure** : describes components and data/control flows between them.
- **Classes** : describes class hierarchy, heritage, interaction.
- **Behaviour** : describes evolution along time in terms of response to sequences of stimuli.

Real-Time systems are reactive

Reactive systems interact with an active environment by reacting to stimuli.



Two kinds of stimuli and reactions :

- Signals : convey a value that changes along time (Ex : temperature, pressure, electric voltage, ...).
- Events : the only information they convey is the instant in time when they occur. They are related to logical rather than numerical variables.

Relationship between stimuli and and reactions

Different occurrences of the same stimulus may provoke different reactions.

Example : Consider a lift control system. pushing the call bottom from the 4th floor provokes reaction "move elevator downwards" if the cabin is in 8th floor and reaction "move the elevator upwards" if the cabin is in the basement.

A reaction depends on a stimulus AND on the internal state of the system.

Behaviour-description languages

Many different languages can be used to describe system behaviour :

- Process algebras.
- Petri Nets.
- State-Transition diagrams.
- ...

We present here the state-transition paradigm for the behavioural description of reactive systems.

Sommaire

The state-transition paradigm

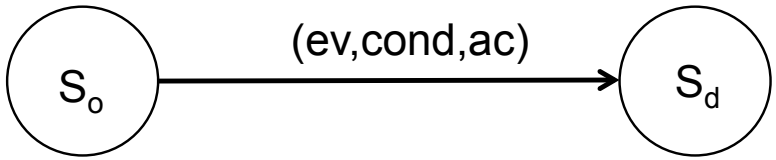
The behavioural model of a system is built by defining,

- A set of states : each state stands for a stable system situation (the elevator is in the 8th floor and there is not any registered service request).
- A set of transitions describing a change of state. Each transition is composed each one of them, of a couple of states (transition origin, transition destination) and a triple (event, condition, action).

.

Graphic representation of state-transition models

State-transition descriptions have a convenient graphic representation : a state is a nodes and a transition is an arrow joining the origin state to the destination state, decorated by the triple (event, condition, action).

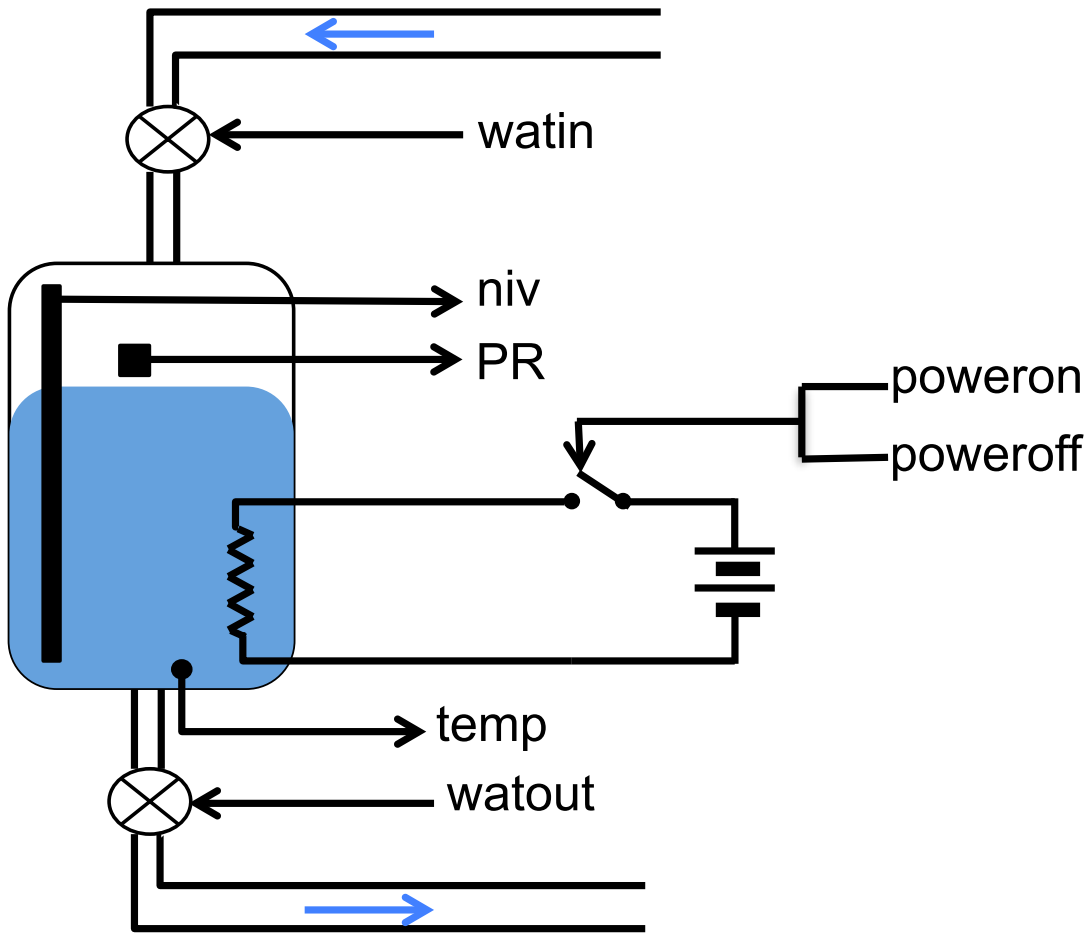


The intended meaning is : if the system is in state S_o and event *ev* occurs then, provided that condition *cond* is satisfied, the system produces reaction *ac* and goes to state S_d.

State-Transition model of a very simple system

A boiler control system

The operating part :



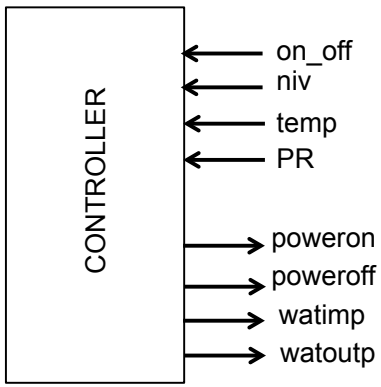
Events:
watin
watout
poweron
poweroff

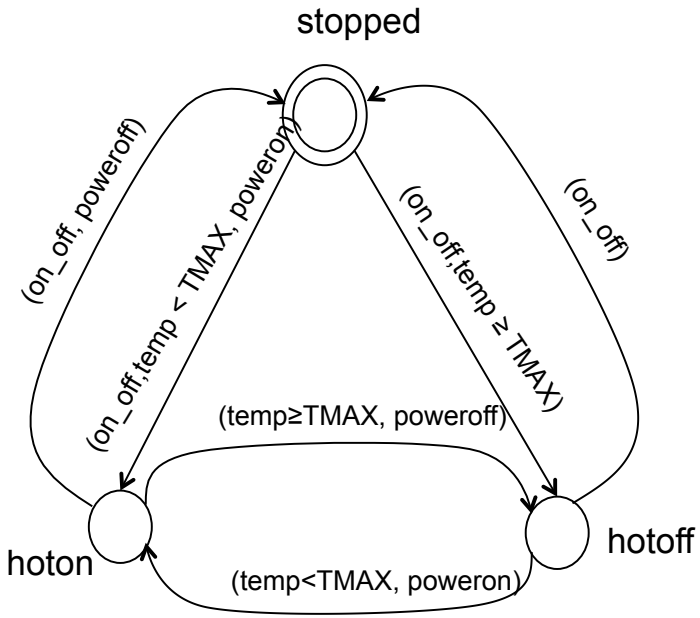
Signals:
temp
niv
PR

State-Transition model of a very simple system

A boiler control system

The controller :





Some remarks

- Transition labels are in general triples $(ev, cond, ac)$ but in practice one or more components of the triple could be lacking (question : can a transition be label-less ?).
- The automaton in the preceding slide presents only the temperature control part. Try to modify it to incorporate water level control. You will rapidly feel the need to be able to introduce structure. Two constructs have been conceived to structure automata models : state hierarchy (Statecharts) or modularity (UPPAAL).

Sommaire

History

UPPAAL results from a collaboration between University of Uppsala (Sweden) and Aalborg (Denmark). The modelling language is based on timed automata [1]. Because of the explicit reference to time in UPPAAL models, this language is adapted to the modelling of Real-Time systems.The first release was in 1995. Currently available free release is 4.1.19. An industrial version of UPPAAL is available (commercial license) and a set of complementary tools has been developed (time conformance analysis, reachability, statistics, test set generation, ...)

1. Alur and Dill, A Theory of Timed automata, Theoretical Computer Science, Elsevier, 1994.

Characteristics of UPPAAL

- Comes as a working environment including an editor, a simulator and a model-checker (verification).
- Structure is given by means of modularity : a model is a set of automaton modules that are instances of parametrized automata templates.
- Both the simulator and the model checker generate execution traces for diagnostic.
- The model-checker verifies properties expressed with temporal logic formulas.

Language components

The modelling language of UPPAAL is graphic with the possibility of adjoining textual elements (functions and procedures, with a language similar to C, ...). The graphic language includes :

- Variables
- States
- Transitions

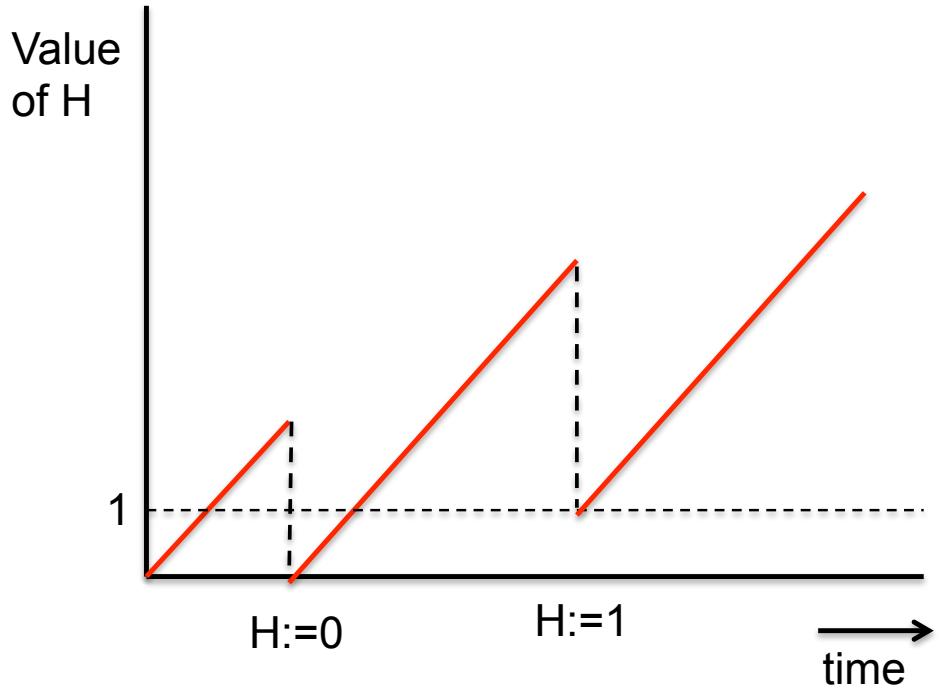
UPPAAL variables

Variables come in simple or array form and,

- have a type : **boolean, integer interval, clock** or **channel**.
- can be local to an automaton or global.
- variable declaration : **bool A[4] ; int [-10,10] N ; clock H1, H2 ; chan CH1 ;**
- clock variables introduce temporal aspects.
- channel variables introduce a synchronisation mechanism based on event emission and reception.

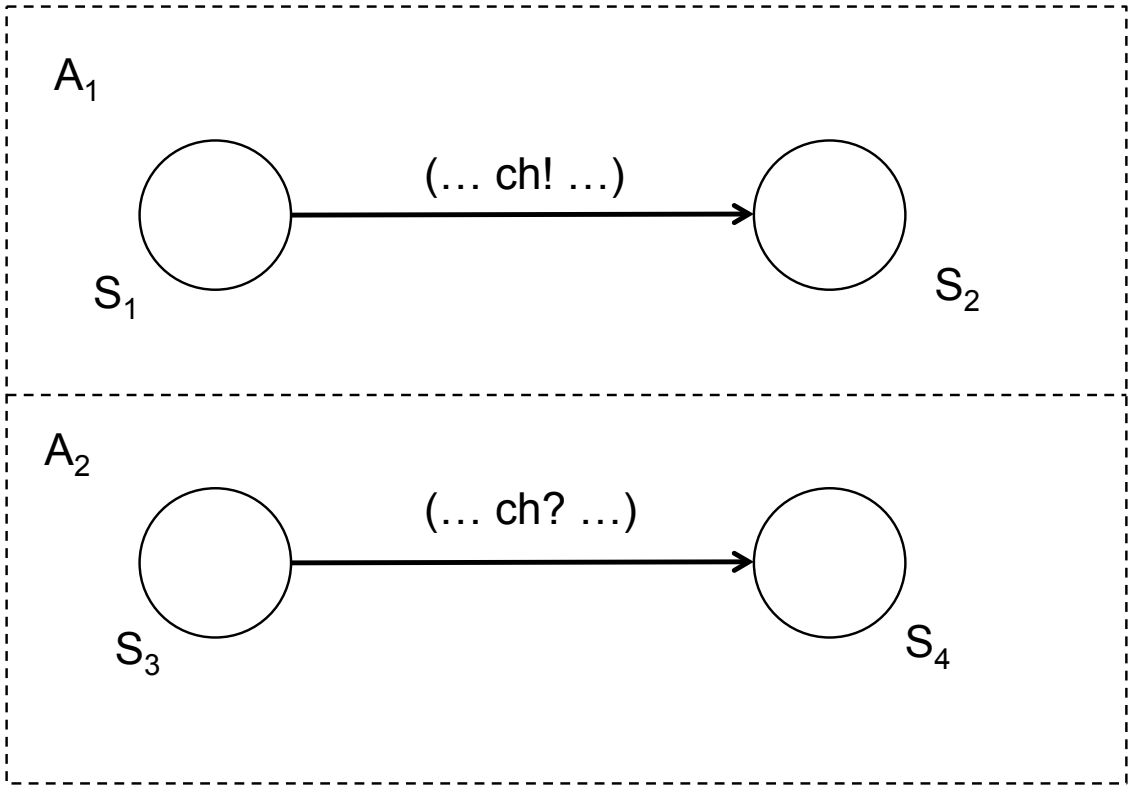
UPPAAL clocks

The value of a clock variable in UPPAAL can be seen as a measure of the time spent from an initial instant T_0. As any ordinary variable, its value can be assigned by an assignment operation at time T but from then on continues to increase linearly. Clock variables can be used in transition conditions and in so called temporal invariants associated to states.



UPPAAL channels

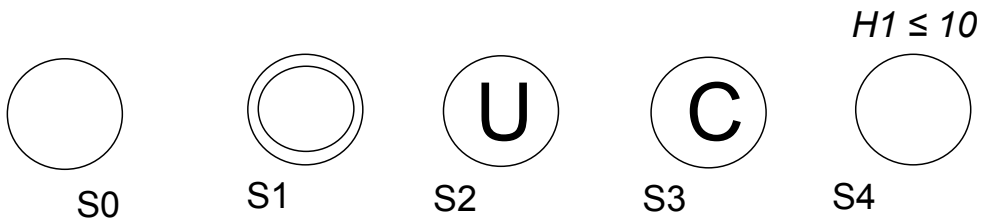
Variables of type channel implement a basic mechanisme to synchronize two or more automata. For automata A_1 an A_2 to be synchronised by channel *ch*, automaton A_1 (A_2) has to include et transition τ_1 with *ch*! in the label of τ_1 and automaton A_2 (A_1) has to include a transition τ_2 with *ch*? in the label of τ_2.



UPPAAL states

States are represented by simple or double circles. A double-circled state(S1) is an initial state (one and only one per automaton). States have a name and a few other attributes :

- a state can be normal (S0), urgent (S2) or committed (S3).
- a state can be assigned a state invariant (S4).

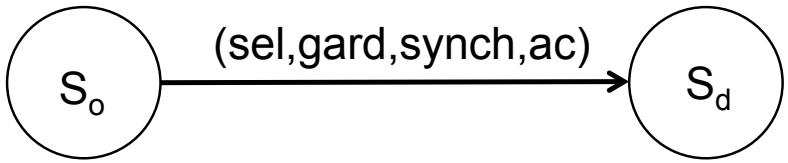


The meaning of urgent states, committed states and state invariants will be explained later.

UPPAAL transitions

As previously stated, transitions are triples (S_o, *label*, S_d), where labels are triples (*ev*, *cond*, *act*). UPPAAL enriches the basic transition label by adding other components. An UPPAAL transition label includes a subset of the following components :

- **Selection** : syntax similar to a declaration (but not an UPPAAL variable declaration) : **i : int [0,3]**.
- **Gard** : this is the transition's condition (a boolean expression).
- **Synchronisation** : one of the expressions **ch !** or **ch ?**, where **ch** is the idetifier of a channel variable.
- **Action** : can be a value assignment (**var = exp**) or a function call (**var = func(...)**) or a procedure call (**proc(...)**)



UPPAAL model definition

The UPPAAL timed model of a reactive system is composed of :

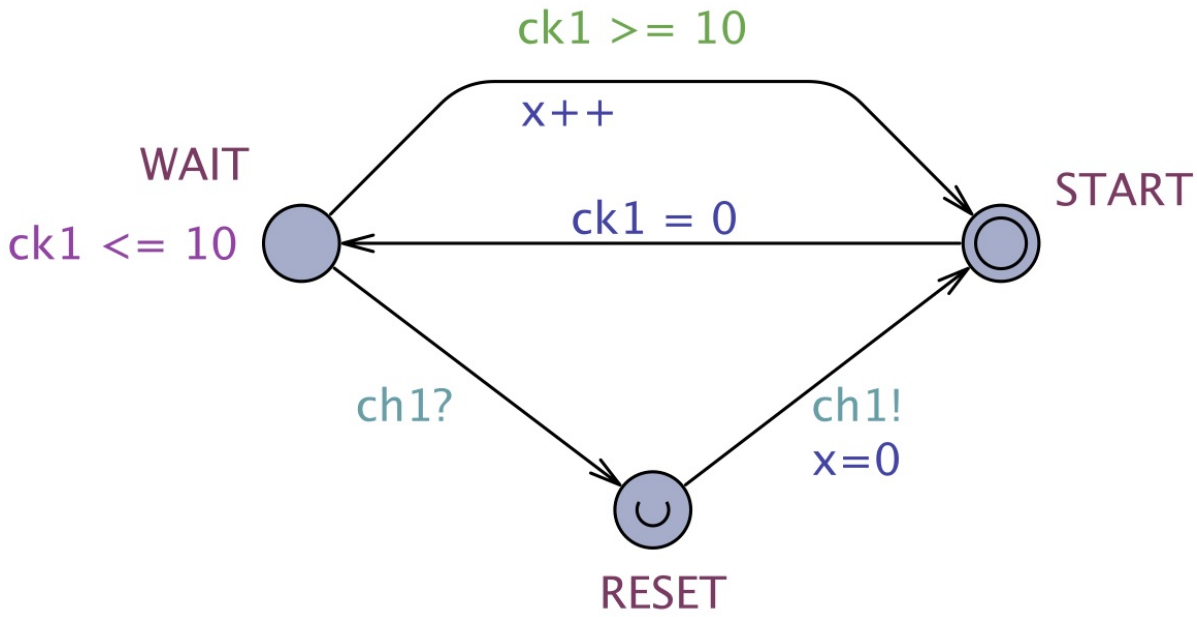
- a set of automaton templates.
- variable declarations local to each template.
- global variables declarations (all channel variables are declared as global).
- the system declaration where the automaton templates are instantiated and the list of component instances is declared.

A simple model : template *S*1

Consider a system composed of two templates, *S*1 and *S*2. Template *S*1 (see below) has the following local declarations :

int [0,10] x ;

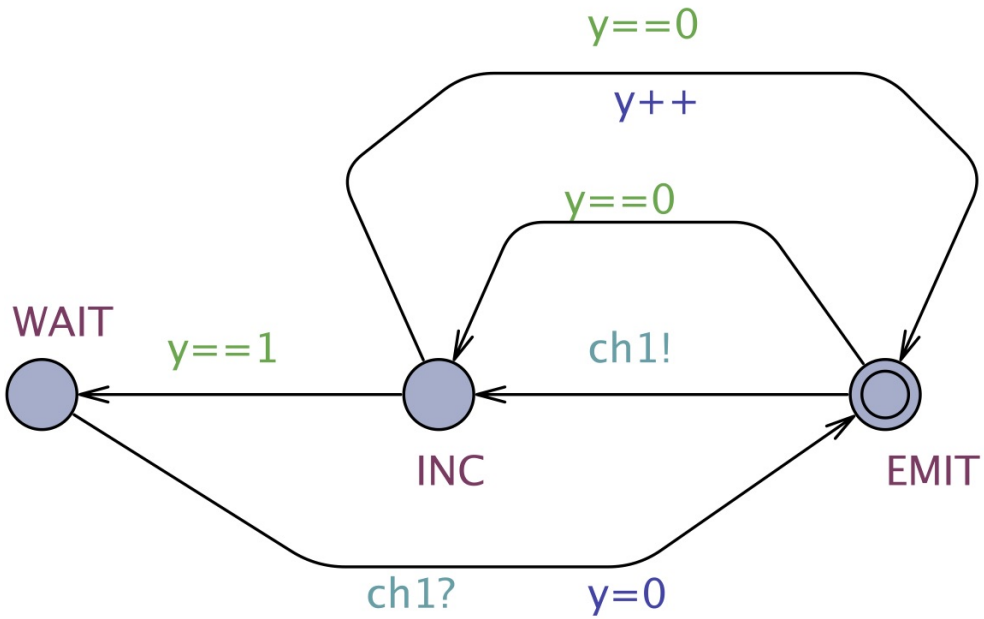
clock ck1 ;



A simple model : template *S2*

Template *S1* (see below) has the following local declarations :

int y ;



A simple model : global and system declarations

- the only global variable is the synchronization channel : **chan ch1 ;**
- the system declaration instantiates *S*1 and *S*2 and integrates them into the system : **system S1, S2 ;**

Note that neither *S*1 nor *S*2 include instance parameters. If they had, before the system declaration, an explicit instantiation should be made.

Sommaire

The behaviour of a system model

Some definitions (1)

Let us consider a system composed of automata $A_1, \cdots, A_n$ such that each automaton A_i has its own set of states. Let V be the set of all variables. The global state of the system at instant i (noted Σ_i) is composed of the local state of each automaton and the value of each variable at that instant.

A transition $\tau = (S_o, (sel, guard, act), S_d)$ without synchronization, is enabled in Σ_i if :

- S_o belongs to Σ_i.
- condition *guard* is satisfied by the value of variables in Σ_i.

The behaviour of a system model

Some definitions (2)

Let us consider a transition $\tau = (S_o, (sel, guard, synch, act), S_d)$ with synchronization, τ is enabled in Σ_i if :

- S_o belongs to Σ_i.
- condition *guard* is satisfied by the value of variables in Σ_i.
- if $synch = ch!$ for some channel variable *ch*, then there is another system's automaton with an enabled transition that has $ch?$ as synchronsation.
- if $synch = ch?$ for some channel variable *ch*, then there is another system's automaton with an enabled transition that has $ch!$ as synchronsation.

The behaviour of a system model

The effect of transition triggering

When a transition without synchronisation or a couple of synchronised transition are triggered, the global state is modified :

- S_o is replaced by S_d for each of the triggered transitions.
- variable values are updated according to the action of each triggered transition.

If two synchronised transitions update the same variable, the updating made by the receiving automaton (*ch*?) prevails.

The behaviour of a system model

Evolutions of a system

An evolution of system model is an ordered sequence $\Sigma_0, \Sigma_1, \cdots, \Sigma_k, \cdots$ such that :

- Σ_0 is the initial global state composed of the initial state of each model's automaton and the initial value of each variable (default or assigned at declaration).
- each Σ_i with $i > 0$ results from the triggering of a transition without synchronisation enabled in Σ_{i-1} or the triggering of two synchronised transitions enabled in Σ_{i-1}.

The behaviour of a system model

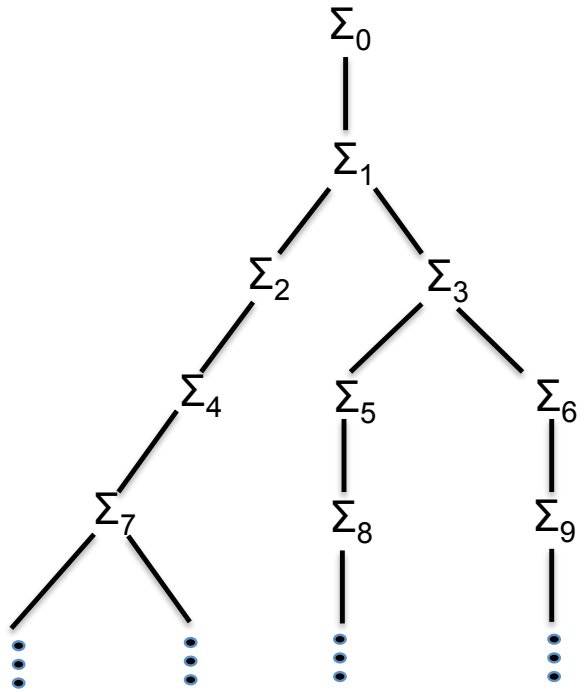
The behaviour of a system model is the set of all possible evolutions of the system, as defined in the preceding slide.

Most of the system models are **non deterministic** : at a given global state more than one transition could be enabled. Depending on the transition that is triggered, different evolutions are performed.

The behaviour of a system model

Tree representation

Because of non-determinism, the behaviour of a system can be represented as a tree (nodes are global states) :



The behaviour of a system model

Open systems and closed systems

Two kinds of systems can be distinguished :

- **Open systems** : react to external signals or events from the environment, with which they interact.
- **Closed systems** : evolutions are provoked only by the internal global state and by internal events (synchronisations).

The behaviour of a system model

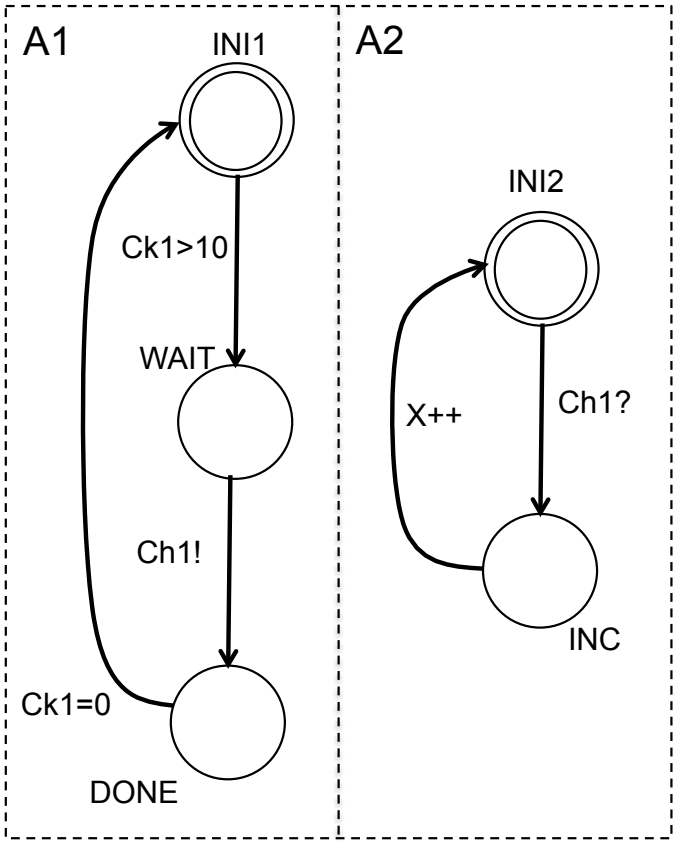
Flow of time and clock updating

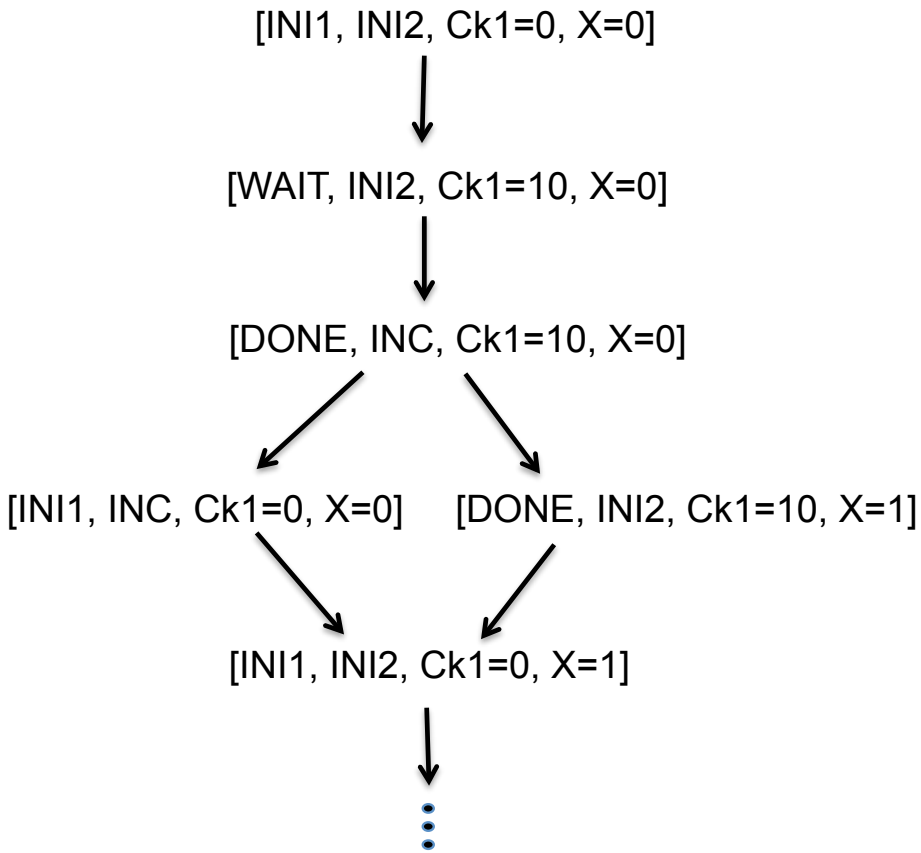
Transition triggering is considered to be instantaneous. Clock values evolve only in global states where there are not enabled transitions. In that situation, the following cases can be considered :

- after some amount of time an external event occurrence or an external signal changes its value, a transition becomes enabled and the global state evolves by the triggering of the transition The clock variables are updated by adding the amount of time that had passed or by transition actions that assign them a new value (open systems).
- after some amount of time a transition becomes enabled and the global state evolves by the triggering of the transition. The clock variables are updated by adding the amount of time that had passed or by transition actions that assign them a new value (open and close systems).
- the flow of time does not produce a situation where at least a transition is enabled. The system model has reached a final global state from which there in not any further evolution (closed systems).

A system and its behaviour

Evolution of time





Variables globales: chan Ch1;
Variables de A1: clock Ck1;
Variables de A2: int X;

The behaviour of a system model

Deadlock situations

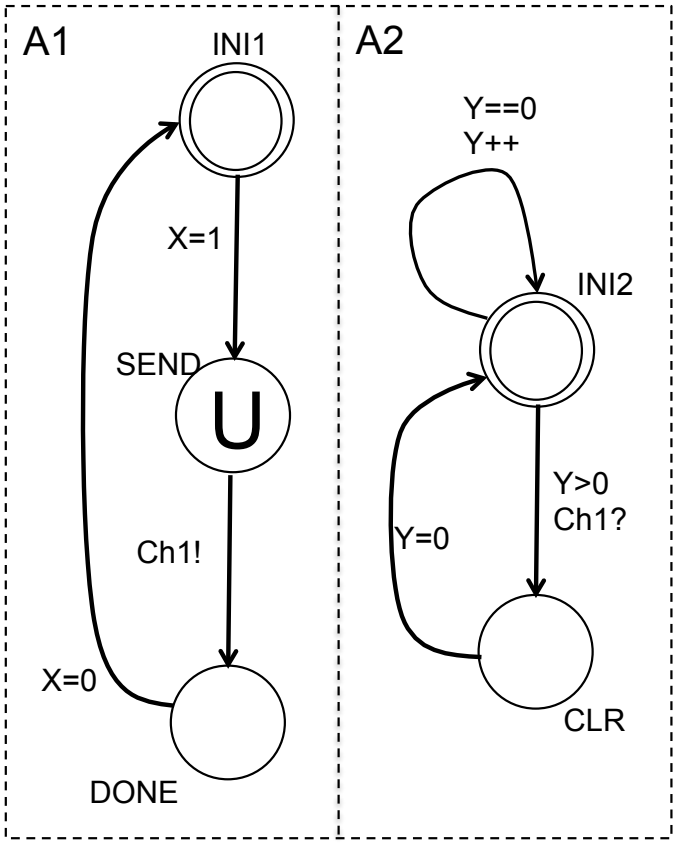
Deadlock situations can arise when urgent or committed states are active. In both kinds of states time is not "allowed" to pass until an outgoing transition is enabled and triggered. The difference between urgent and committed states is the following :

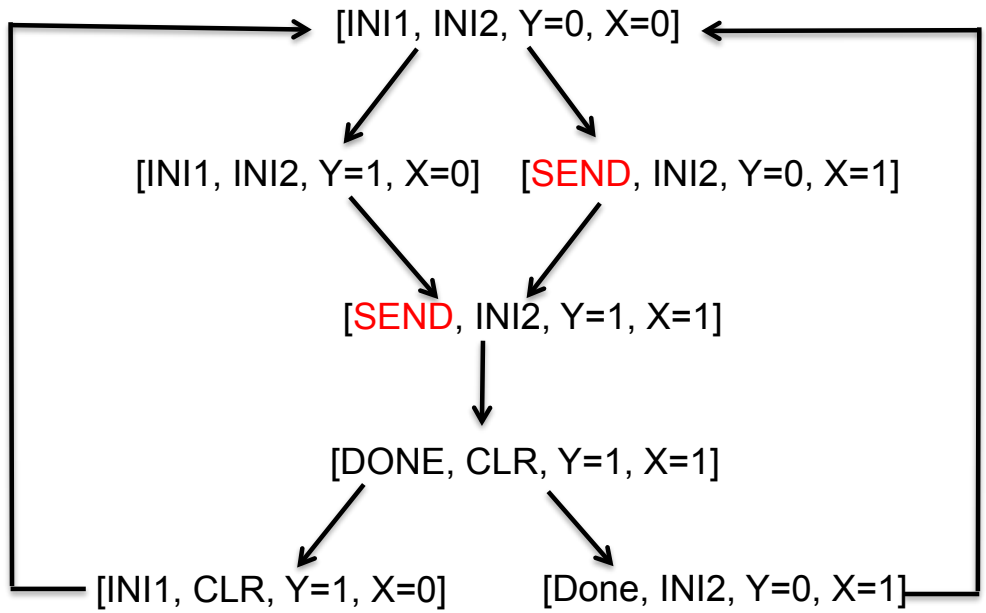
- if an urgent state becomes active it must be abandoned immediately by the triggering of an outgoing transition τ but other instantaneous transitions can be triggered before τ.
- if a committed state becomes active it must be abandoned immediately by the triggering of an outgoing transition τ and not any other transition is allowed to be triggered before τ.
- if a state with an invariant assigned to it must be abandoned as soon as its invariant evaluates tu "false". We can say that when the invariant becomes false, the state becomes a committed state.

If the conditions presented here are not satisfied, then a **deadlock** situation arises.

A system and its behaviour

A case without deadlock

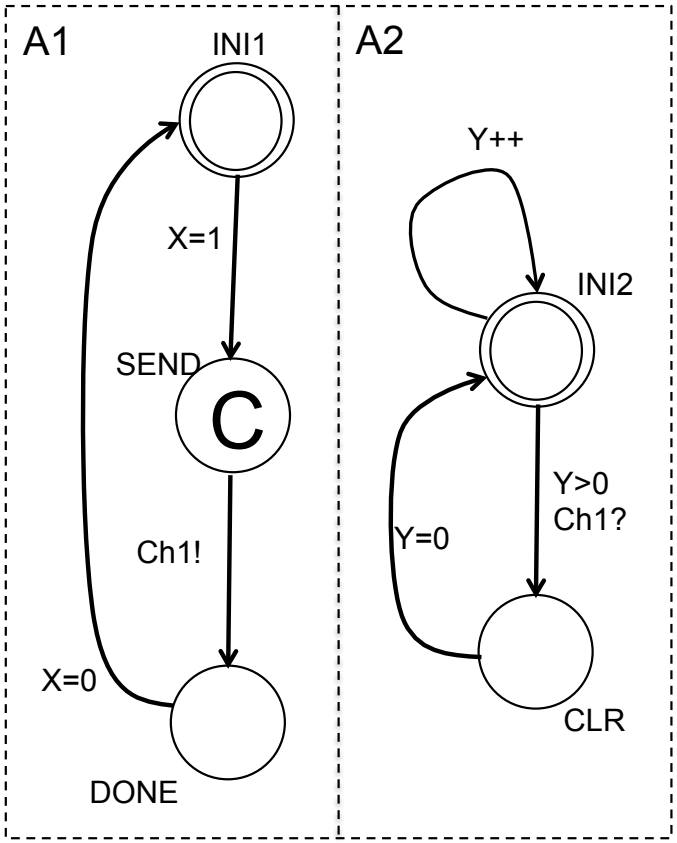


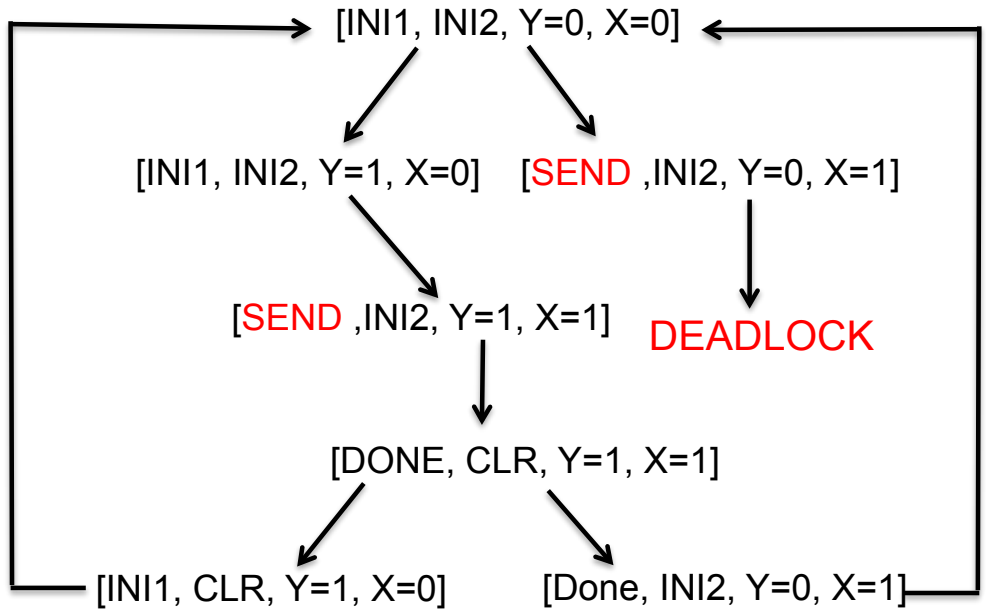


Variables globales: chan Ch1;
Variables de A1: int X;
Variables de A2: int Y;

A system and its behaviour

A case with deadlock due to a committed state

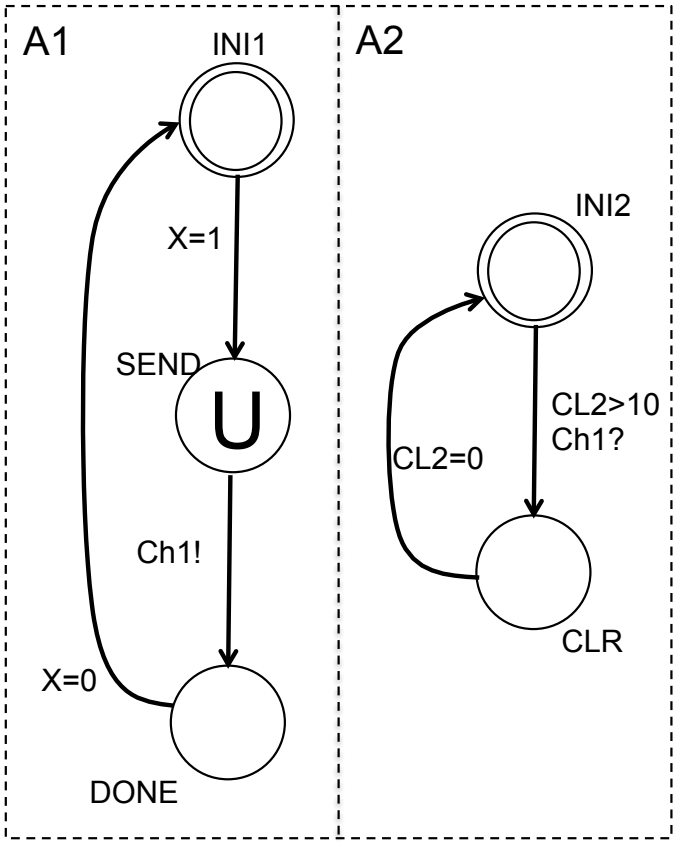




Variables globales: chan Ch1;
Variables de A1: int X;
Variables de A2: int Y;

A system and its behaviour

A case with deadlock due to an urgent state



[INI1, INI2, Y=0, X=0]

↓

[INI1, SEND, Y=0, X=1]

↓

DEADLOCK

Variables globales: chan Ch1;
Variables de A1: int X;
Variables de A2: clock CL2;

Sommaire

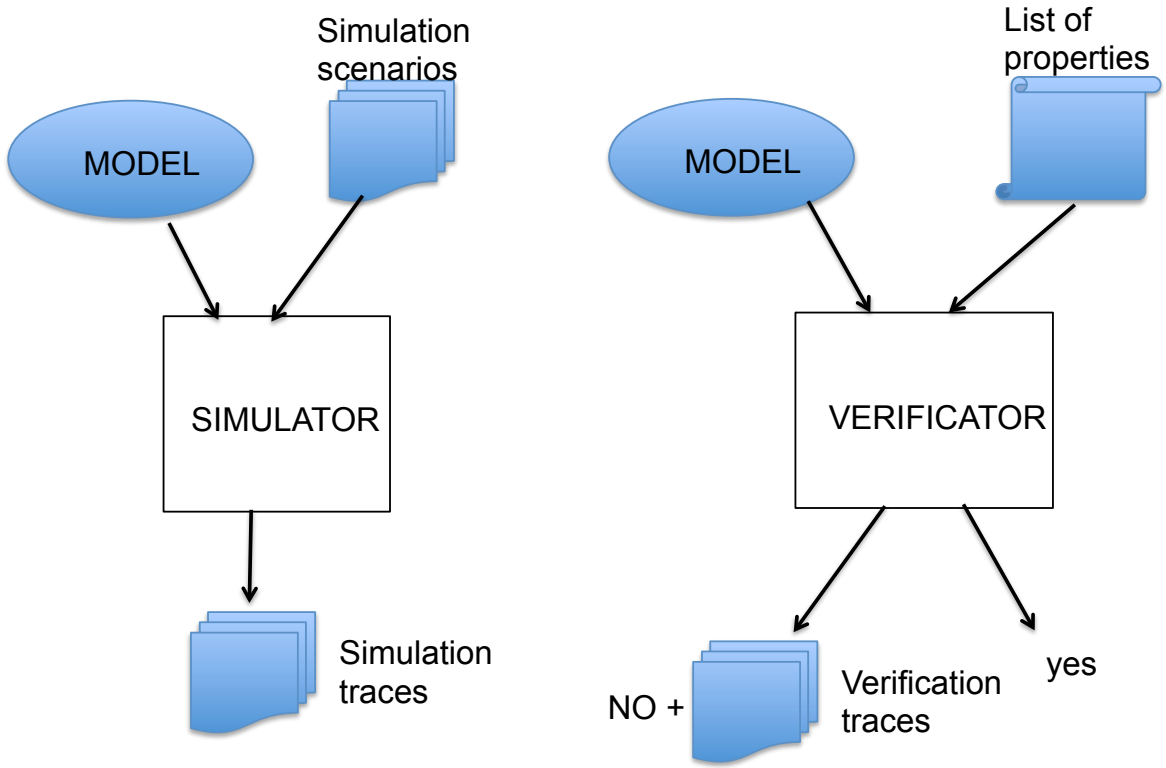
Exploiting models during the design process

Models are built during the design process of a reactive system but, what is the use of those models ?

They can be used to increase the confidence in the good quality of the design and, more importantly, to prove that system, as it is being designed, satisfies a set of **safety** properties. There are two ways to exploit the system models :

- **simulation** : the model can be "executed" to obtain its response to a simulation scenario, registered in a simulation trace. The quality of simulation depends on the set of simulation scenarios.
- **verification** : a property P that the designed system is supposed to satisfy is formulated and the model is "analysed" by a verification program to determine if the model satisfies P or not.

Organisation of simulation and verification procedures



Verification techniques

Verification tools base on one of the following algorithmic approaches :

- **semi-automatic deduction :** based on algorithms of theorem-proving. The model is transformed to a formal logic representation and the verification tool tries to "deduce" the property from the model (seen as a set of premises). Example of deduction-based toolbox : the B method (www.methode-b.com).
- **model-checking** : based on decision algorithms(logical consequence). The behaviour of the system is explored to determine if the property is satisfied by the model. The verification tool of UPPAAL is based on model-checking.

Verification techniques

Pros and cons

Each approach has its own advantages and disadvantages :

- **semi-automatic deduction :** cannot be fully automatic. Frequently requires human intervention with high technicality.
- **model-checking** : is simple ("push-button" style) but the model-checking program not always terminates (combinatoric explosion of the state space [2]).

2. The term "state space" is used to refer to the behaviour of the system, i.e. the set of all possible evolutions.

Sommaire

Properties of system behaviour

Properties that refer to the behaviour of a system express something that is false or true of,

- **An isolated global state :** expressed with the language of first order predicates.
- **a single evolution** : expressed with the language of first order predicates + linear temporal operators.
- **all or some of the evolutions** : from the root of the behaviour tree expressed with the language of first order predicates + linear temporal operators + tree temporal operators.

Temporal logic operators

Linear temporal operators :

- **necessity** : all the states of an evolution must satisfy state property P (is writen $[]P$).
- **eventually** : sooner or later some state in the evolution satisfies state property P (is writen $<>P$).

Tree temporal operators :

- **All** : all the evolutions of a behaviour tree must satisfy linear property P (is writen $\mathbf{A}P$).
- **some** : some evolutions of a behaviour tree must satisfy linear property P (is writen $\mathbf{E}P$).

Some interesting types of properties

and their expression with the UPPAAL property language

- **Safety** "something wrong must not occur" : if state formula W expresses something wrong then safety is expressed by **A[] not** W. This is also called "invariance".
- **Weak reachability** "from the root state at least an evolution leads to a state that satisfies state property P. Expressed by **E<>**P.
- **Consequence** "if a property P_1 is satisfied in some state then another property P_2 will be satisfied later". Expressed by **A**$(P_1 \Longrightarrow <> P_2)$. UPPAAL accepts a simplified syntax : $P_1 \ ==> \ P_2$