Chapter x
Verification of specific properties

Serge HADDAD , François VERNADAT[1]

1. Introduction

In the previous chapter of this book of the Petri nets [DIA 01], we studied the checking of generic properties of Petri nets such as boundness or liveness [HV 01]. If these properties inform the designer about the general behavior of the net, those must be supplemented by the analysis of the specific properties of the modelled system. Also we lean in this chapter on the expression and the checking of specific properties of the Petri nets.

Generally, the designer of an application defines the functions and/or the services of this one through a specification. Once its modelled application, it wishes to check that its model conforms to the specification. In order to develop algorithms and tools for this checking, it is necessary to formalize the concept of specification. Two possibilities were largely studied with this goal : either the specification is defined by a set of formulas of an adequate logic, or the specification is defined using a model of behavior. We will thus explore these two ways which in practice are complementaries : certain properties will be expressed more easily using formulas and others more easily using a behavior. To fix the ideas, let us consider a simplified example of ressource allocation in mutual exclusion : 2 customers are in competition to reach a resource. The access control policy is ensured by a mechanism whose we will make abstraction.

[1]LSV, École Normale Supérieure de Cachan 61, avenue du Président Wilson 94235 CACHAN Cedex - France (haddad@lsv.ens-cachan.fr)
LAAS-CNRS, 7, Avenue du Colonel Roche F-31077 Toulouse cedex, (francois.vernadat@laas.fr)

Among the properties to be checked, we must express the property of mutual exclusion : P1 "the resource is used with more by one customer", and by analogy with the problems of the philosophers, which one generally names the absence of starvation : P2 "A customer awaiting the resource will obtain it in a finite time ". One wants to also be able to specify the operation of a customer and to express P3 "the customer sends initially a request to obtain the resource (A), that it receives then an agreement of use (b) and that finally (c) it sends a message of release before turning over in its initial state".

P1 and P2 will be expressed simply using formulas of temporal logics while P3 will be expressed in a more compact way using a behavior such as that represented on the left of the figure 1.

A contrario, P1 will be expressed only one very indirect by the behavior represented on the straight lines of the figure 1 : between does two consecutive entries in critical section (event ? *Ack*), is inevitably the exit of the only customer present in critical section (event ! *Rel*).

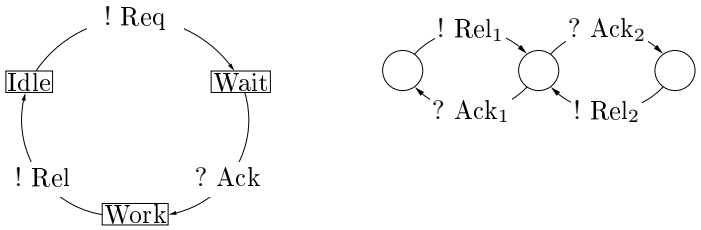


Figure 1: Examples of behavioural specifications

A logic ready to reason on the behavior of dynamic systems to discrete events must necessarily integrate the concept of sequence of states (finished or infinite) correspondent into a possible execution. Moreover it must be able to express properties of safety like "To more the one process in the course of execution of a critical section, in any state of the sequence" (cf P1), of the properties of liveness like "If in a state, a process requires to carry out a a critical section then in a future state this process will carry out this critical section" (cf P2) and of the properties of equity as "Any process able to be carried out in an infinity of states will be chosen by the scheduler an infinity of time". The key concept is here the time seen like a discrete succession of moments and logics which integrate this concept are called underline logical temporal.

These logics are distinguished according to two axes. Parallelism and/or the non-determinism imply the existence of various executions of the same system and require their simultaneous taking into account. Then :

- the whole of the executions is represented as a tree where the various successors of a state are obtained by the possible instances of events in this state. One then speaks about *branching logical temporal*.
- the whole of the executions may be also represented as a set of execution sequences. One then speaks about *linear logical temporal*.

The second axis relates to the elements of the sequence.

- We can consider a sequence of states characterized by a their set of atomic propositions. One then speaks about *state-based temporal logic*.
- We can also consider a sequence of elementary transitions, each one labelled by an event. One then speaks about *event-based temporal logic*.

Also in the first part, we will introduce the syntax and the semantics of a propositional branching time logic called $\mathcal{CTL}^*$. We will present two very studied fragments of them $\mathcal{CTL}$ and $\mathcal{LTL}$. We will show then how to carry out the checking of formulas on finite states models.

We will complete this section by indicating the adaptations to be taken into account within the framework of the Petri nets. The principal point is to consider in a suitable way the various types of sequence of firing (finished, maximum finished or infinite).

After having seen the logical approach for the checking, we will be interested in the "behavioral" approach. The logical approach is sometimes described as "double model" in the sense where one has a logic for to specify the properties to be checked and of one structure which represents the behavior of the system (the "Kripke'structure" which is defined by the reachable markings graph in the case of a system describes by a Petri net). At the opposite, the behavioral approach is sometimes described as "simple model" in the sense where we only dispose of a single structure, a "labelled transitions system" (a structure close to the the reachable markings graph for Petri Nets. This one makes it possible to represent at the same time the behavior system and its specification.

The behavioral approach proceeds by comparison. Using various relations of equivalences or pre-orders, two are compared behaviours : those satisfy the same properties if and only if they are equivalent. Various behavioural equivalences have been introduced to take in account several classes of properties or, in an equivalent way, several points of view to consider when a system is analyzed. Among these different point of view, we will find the taking into account of parallelism and non-determinism. As for temporal logics, we will be brought to distinguish two great families from relations of behavioural equivalences : the family of the "equivalences of traces" which consider the execution of a system through the set of its sequences of execution (cf linear temporal logics) and the family of "bisimulations" which consider the execution of a system through its "tree" of execution (cf branching time temporal logics). For these two families, one can also be brought to privilege the "states" of an execution (cf state-based

temporal logics) or the events which constitute the execution (cf event-based temporal logics).

A first section will enable us to introduce in an unformal way various possible points of view when the behavior of two systems is compared. The second section will present the concept of bisimulation and simulation. The associated procedures of decision will be presented. The third part will present "weak" equivalences which make it possible to compare systems described at various levels of abstraction. The last section will attempt to show the links between the behavioral approach and the logical approach : we will present logic $\underline{\mathcal{HML}}$ [HM 85] which gives a modal characterization of the bisimulation. relation. In the other direction, we will have the results of [BCG 91] which gives a behavioural characterization of temporal logic $\mathcal{CTL}^*$.

In the last part, we will analyze the decidability of the evaluation of formulas of temporal logic on a Petri net and the test of bisimulation of a net marked with a labelled transitions system. More precisely, we will establish as within the framework of a propositional temporal logic, the evaluation is undecidable as well for the fragment $\mathcal{CTL}$ as for the fragment ltl. This result also holds for event-based arborescent logic. In these three cases, the formulas used require only one number very limited temporal operators what indicates the robustness of the result (see for example [ESP 98]). While being based on similar arguments, one will show that the test of bisimulation of two marked nets is also undecidable [JAN 95].

Contrary for an event-based linear logic temporal very expressive (the linear μ-calcul), the evaluation of formulas remains decidable [ESP 97]. In the case of maximal sequences, the procedure is based on the decidability of the reachability [MAY 84] while for the infinite sequences, one is reduced to the technique shorter sequences seen to chapter 4 of the first treating volume of the Petri nets [HAD 01] or also used in [RAC 78, YEN 92]. Finally the test of bisimulation of a marked net and a finite transitions system becomes decidable (here still using the test of accessibility) [JAN 99]. This result is all the more interesting as very often the specification of a service is given by such a system and the validation consists in comparing this specification with the Petri net who implements it.

2. Kripke's Structures and transitions systems

The labelled Kripke's structures describe in a sufficiently generic way the behavior of the systems which one wishes to study. Those consist of a set of states for which certain propositions are checked and of a set of binary relations between states, subscripted by the events of the system.

Definition 1 *A labelled Kripke's structure* $\mathcal{LKS} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$ *is defined by :*

- AP *is a set of atomic proposals*
- Σ *is a finite alphabet of events*
- S *is a set of states*
- $\xrightarrow{a}$ *is a binary relation* $\subset S \times S$
- $\nu : S \rightarrow 2^{AP}$ *is a labelling which associates in each state,* $\nu(S)$ *the set of the atomic propositions holding in* s.

When one studies a structure of Kripke labelled, one considers it generally provided with an initial state s_0 what one notes by (SKE, s_0). When one disregards event, one then has business with a structure of Kripke. Contrary, if one disregards atomic proposal, one speaks about system to transitions labelled. The two following definitions formalize these concepts.

Definition 2 *A structure of Kripke* $\mathcal{KS} = \langle AP, S, \rightarrow, \nu\rangle$ *is defined by :*

- AP *is a set of atomic proposals*
- S *is a set of states*
- $\rightarrow$ *is a binary relation* $\subset S \times S$
- $\nu : S \rightarrow 2^{AP}$ *is a labelling which associates in each state,* $\nu(S)$ *the whole of the atomic propositions holding in* s.

Definition 3 *A labelled transitions system* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *is defined by :*

- Σ *is a finite alphabet of events*
- S *is a set of states*
- $\xrightarrow{a}$ *is a binary relation* $\subset S \times S$

Thereafter, we will note $s \xrightarrow{a} s'$ to indicate that $(s, a, s') \in S \times \Sigma \times S$. By abuse notation, we will also note for $\sigma \in \Sigma^*$: $s \xrightarrow{\sigma} s'$ to indicate that s is accessible starting from s via the sequence from actions (the word) σ.
the systems which we consider being able to be not-determinists, we will note for $s \in S, E \subset S$ and $a \in \Sigma$: $s \xrightarrow{a} E\sigma E = \{s' \in S : s \xrightarrow{s} s'\}$
We will note finally $s \overset{a}{\not\rightarrow}$, to indicate that s does not have a successor by the action a and $s \not\rightarrow$ to indicate that s does not have any successor (i.e., constitutes a state of blocking).

These structures can they be also initialized.

3. Temporal Logic

3.1. *Syntax and Semantics*

Since one wishes to check dynamic systems with discrete events, let us express what is common to all these systems : states and the reachability relation between states. As example, a state of a distributed application is characterized by the state of the processes (value of the variables, instruction counter, ...) and the state of the environment (e.g messages of the channels). In front of the diversity of the possible representations, one will be satisfied of a largely sufficient abstraction in the majority of the cases to knowing a set of atomic proposals (noted $P, Q, \ldots$). Starting from a given state, the relation of succession induces a set (generally infinite) of sequences of states begin with this state, still called paths in the terminology of temporal logic. Also propositional arborescent logic that we will study $\mathcal{CTL}^*$ defines it inductively by a syntax of formulas of state and of path [EME 96].

Definition 4 (Syntax of $\mathcal{CTL}^*$) *Let AP be a set of atomic proposition, then the formulas of $\mathcal{CTL}^*$ are defined by the following rules :*

- *S1 Each atomic proposition P is a state formula.*
- *S2 If f and g are state formulas then f **AND** g and **NOT** f are state formulas.*
- *S3 If f is a path formula then **E** f and **A** f are state formulas.*
- *P1 Each formula of state is a path formula.*
- *P2 If f and g are path formulas then f **AND** g and **NOT** f are path formulas.*
- *P3 If f and g are formulas of way then **X** f and f **U** g is formulas of way.*

Only the rules $S3$, $P1$ and $P3$ require explanations. One wishes to reason on the sequences resulting from a state. Thus **E** f is checked if starting from this state there exists a sequence which checks f. **A** f is checked if starting from this state all the sequences check f. If f is a formula of state then f also interprets as a formula of way which is evaluated on the first state of the sequence. **X** f (**X** for "next") consists in evaluating f on the private under-sequence of the first state. Finally f **U** g (**U** for "until") is checked if there exists a suffix of the sequence for which g is checked and such as all the preceding suffixes (including the initial sequence) check f. In other words, f remains checked until g is checked and g will be it. We formalize now the semantics of $\mathcal{CTL}^*$ by introducing the concept of model and satisfaction of formula by a model.

Definition 5 (Model of $\mathcal{CTL}^*$) *A model of $\mathcal{CTL}^*$ is a Kripke's structure $\mathcal{KS} = \langle AP, S, \rightarrow, \nu \rangle$ such as $\rightarrow$ is a total binary relation : $\forall S \in S, \exists T \in S$ such as $s \rightarrow t$*

Traditionally temporal logic reasons on infinite sequences. Indeed, this one is interested particularly in properties of equity which have meaning only in this context. This explains the constraint on the relation $\rightarrow$. Also a sequence $\sigma = (s_0, s_1, \ldots)$ is an infinite sequence of states such as $\forall I \in \mathbb{N}, s_i \rightarrow s_{i+1}$. We will reconsider later this constraint in the context of the Petri nets. The sequence σ^i indicates the suffix of σ, $(s_i, s_{i+1}, \ldots)$ from where $\sigma^0 = \sigma$.

Let us note $\overline{AP} = \{\mathbf{NOT} P \mid P \in AP\}$. To simplify, we will consider in the sequel that the labelling function ν takes values in $2^{AP \cup \overline{AP}}$ with the obvious constraint that :
$\forall P, s \; |\{P, \mathbf{NOT}\ P\} \cap \nu(s)| = 1$ (an atomic proposition is either true or false)

Definition 6 (Semantic of $\mathcal{CTL}^*$) *Let $\mathcal{KS}$ be a model, s a state of SK and $\sigma = (s_0, s_1, \ldots)$ a sequence of $\mathcal{KS}$, then the satisfaction of a formula of $\mathcal{CTL}^*$ on this model is defined par :*

S1 $\mathcal{KS}, s_0 \models P$ *if and only if* $P \in \nu(s_0)$.

S2 $\mathcal{KS}, s_0 \models F \boldsymbol{AND} g$ *if and only if* $\mathcal{KS}, s_0 \models f$ *and* $\mathcal{KS}, s_0 \models g$.
$\mathcal{KS}, s_0 \models \boldsymbol{NOT} f$ *if and only if there are not* $\mathcal{KS}, s_0 \models f$.

S3 $\mathcal{KS}, s_0 \models \boldsymbol{E} f$ *if and only if* $\exists \sigma$ *resulting from* s_0 *such as* $\mathcal{KS}, \sigma \models f$.
$\mathcal{KS}, s_0 \models \boldsymbol{A} f$ *if and only if* $\forall \sigma$ *resulting from* s_0, $\mathcal{KS}, \sigma \models f$.

P1 *Is f a formula of state,* $\mathcal{KS}, \sigma \models f$ *if and only if* $\mathcal{KS}, s_0 \models f$.

P2 $\mathcal{KS}, \sigma \models F \boldsymbol{AND} g$ *if and only if* $\mathcal{KS}, \sigma \models f$ *and* $\mathcal{KS}, \sigma \models g$.
$\mathcal{KS}, \sigma \models \boldsymbol{NOT} f$ *if and only if* $NOT(\mathcal{KS}, \sigma \models f)$.

P3 $\mathcal{KS}, \sigma \models F \boldsymbol{U} g$ *if and only if* $\exists i$ *such as* $\mathcal{KS}, \sigma^i \models g$ *and* $\forall j$ *such as* $pr \notin PROM(e_m)$

In practice, $\mathcal{CTL}^*$is enriched by abbreviations which simplify the expression of the properties :

(OR) f **OR** $g \equiv$ **NOT** (**NOT** f **AND NOT** g)
(true) $true \equiv$ **NOT** P **OR** P
(false) $false \equiv$ **NOT** $true$
(F) $\mathbf{F} f \equiv true \mathbf{U} f$
(G) $\mathbf{G} f \equiv$ **NOT** **F** **NOT** f
(W) $f \mathbf{W} g \equiv f \mathbf{U} g$ **OR** $\mathbf{G} f$

F f means that f will be true for a suffix of the considered sequence. **G** f means that f is true for all the suffixes of the sequence considered. Contrary to

f **U** g, f **W** g (**W** for "weak until") does not imply that g is true for a suffix. In this case, f remains true for all the suffixes. As example, $\mathbf{G}F \Leftrightarrow F$ **W** $false$.

$\mathcal{CTL}^*$ is a very expressive language. In order to obtain effective algorithms of evaluation, one is led to restrict this language. The two most significant restrictions are $\mathcal{CTL}$ and $\mathcal{LTL}$.

$\mathcal{CTL}$ is the language formed of the syntactic rules $S1$, $S2$, $S3$ and $P0$:

P0 If f and g are state formulas then **X** f and f **U** g is path formulas.

$\mathcal{CTL}$ is focused on the concept of state. Indeed, one can entirely describe syntax without defining the path formulas using the four operators $\mathbf{AX}f$ (for any state successor of the state considered, f holds), $\mathbf{EX}f$ (there is a state successor of the state considered for which f holds), $\mathbf{A}F\mathbf{U}g$ (for any sequence resulting from the state considered, f holds until g holds and g will be true) and $\mathbf{E}F\mathbf{U}g$ (there exist a sequence resulting from the state considered such as f holds until g and g will be true). The interest of $\mathcal{CTL}$ lies in the fact that, on the one hand, it is sufficiently expressive for the specification of the majority of the usual properties and that, on the other hand, the methods of checking of the satisfaction of a formula by a model have a complexity proportional to the size of the model and the size of the formula. However certain properties of fairness are not expressible in $\mathcal{CTL}$. This gap led to various extensions of the model by operators such as $\mathbf{AGF}f$ (for any sequence resulting from the state considered, f holds an infinite number of states of the sequence) which make it possible to express usual concepts of fairness. These extensions also have methods of verification of polynomial complexity. We left the reader to the references which follow for more precise details on this langage [EC 81, EC 82, EH 85].

Example
The formula **AG** (**A** *req* **U** *serv*) expresses that starting from any state which contains a request, in any sequence the request will be present until it is served.

$\mathcal{LTL}$, the language formed of the syntactic rules $S1$, $P1$, $P2$ and $P3$ is focused on the concept of sequence. Indeed, one can entirely describe syntax without defining the state formulas by considering that the atomic propositions are path formulas to be checked on the first state of the sequence.

The use of such a logic is justified when you consider the point of view of an observer which cannot interact with the system. In this case, only the sequences are significant One of the interests of $\mathcal{LTL}$, illustrated by the following chapters is its applicability with partial order techniques for the reduction of complexity of the checking. Generally a model $\mathcal{KS}$ is initialized by a state s_0 and the satisfaction of the formulas is evaluated on $\mathcal{KS}, s_0$. As being given a formula $\mathcal{LTL}$ path f, one will note by abuse of notation $\mathcal{KS}, s_0 \models f$ to indicate $\mathcal{KS}, s_0 \models \mathbf{A}f$.

Example
The formula **GF** *p.exec* **OR FG** *p.bloq* expresses that during any execution either the process p is indefinitely blocked starting from a given state, or this process is selected an infinity of time by the scheduler.

3.2. *Methods evaluation*

The objective of a method evaluation is to check if a formula is satisfied by a particular model. In this paragraph, we treat only finite models. The study of the verification of the infinite models such as the reachability graphs of unbounded Petri nets will be done at the end of the chapter.

In the sequel $\mathcal{KS}$ will denote the model, f_0 the formula to be checked and s_0 the initial state of the model. The problem to be solved will be to determine if $\mathcal{KS}, s_0 \models f_0$ holds.

3.2.1. *Checking of formulas* $\mathcal{CTL}^*$

First we show that if we dispose of an evaluation method for $\mathcal{LTL}$ one can build a method evaluation of $\mathcal{CTL}^*$. The principle of construction is relatively simple. First First we eliminate the operator **E** by replacing it by the equivalent expression **NOT A NOT**. Let us consider the syntactic tree of a formula of state f_0 of $\mathcal{CTL}^*$:

- a node labelled by **A** which does not comprise in its sub-tree this same operator **A** prefix g a formula of $\mathcal{LTL}$.
- We then evaluate g for all the states of the model and we create a new proposition $[\mathbf{A}G]$. This proposition is assigned to the states of the model according to the result of the evaluation of g.
- We substitute in f_0, $\mathbf{A}g$ by $[\mathbf{A}G]$ and we iterate the process until the disappearance of the operator **A**.
- the formula obtained is then a formula of propositional logic which is evaluated locally on each state.

We call " $\mathcal{CTL}^*$-checks" the method of required checking and "$\mathcal{LTL}$-checks" the method of checking of formulas of $\mathcal{LTL}$. The text of the method is given below.

$\mathcal{CTL}^*$-checks $(\mathcal{KS}, s_0, f_0)$
While $\exists f = \mathbf{A}g$ subformula of f_0 where $f \in \mathcal{LTL}$ **Do**
  Introduce a new atomic proposition $[f]$
  For each state s **Do**
    If $\mathcal{LTL}$-checks $(\mathcal{KS}, S, g)$ **Then**

add $[f]$ to $\nu(s)$
Else
add **NOT**$[f]$ to $\nu(s)$
End if
End for
Substitute $[f]$ to f in f_0
End While
// f_0 is now a propositional logic formula
If $s_0 \models f_0$ **Then**
return(TRUE) ;
If not
return (FALSE) ;
End if

We apply this method to the formula **A**(**FAG** P **AND G AF**Q) **AND** R :
- **AG**P is a formula of the required type.
- One evaluates on each state **G**P and one updates in consequence [**AG** P].
- One transforms the initial formula which becomes :
  A(**F**[**AG** P] **AND G AF** Q) **AND** R.
- Then the formula is transformed into **A**(**F**[**AG**P]**ANDG**[**AF**Q])**AND**R.
- the final formula is a propositional formula
  [**A**(**F**[**AG** P] **AND G**[**AF** Q])] **AND** R.

3.2.2. *Verification of $\mathcal{LTL}$ formulas*

We examine now the verification of $\mathcal{LTL}$ formulas. We will proceed into three steps :
- We "normalize" the formula so as to push back the operator **NOT** in front of the atomic propositions.
- We define the automata with promises which accept infinite sequences of a model. Then we exhibit a construction of an automaton which accepts exactly the sequences which check a given formula.
- Being given an initialized model, we show how to check that this model comprises at least a sequence accepted by a given automaton.

The verification method then consists to build the automaton associated with **NOT**f_0 and to check that the model $(\mathcal{KS}, s_0)$ does not comprise a sequence accepted by this automaton.

Normalization of a $\mathcal{LTL}$ formula

We normalize the formula using operators **OR** and **W** ("weak until"). The normaliization of a formula f noted $norm(F)$ pushes back the operator **NOT** in front of the atomic propositions. The following equivalences are easy to check

starting from the definitions
(e.g. **NOT** $(f$ **U** $g) \Leftrightarrow$ (**NOT** g **AND** $f)$ **W** (**NOT** f **AND NOT** $g)$).

- $norm(P) = P$, $norm($**NOT** $P) =$ **NOT** P, $norm(\mathbf{X}f) = \mathbf{X}norm(f)$
- $norm(f$ **OR** $g) = norm(f)$ **OR** $norm(g)$
- $norm(f$ **AND** $g) = norm(f)$ **AND** $norm(g)$
- $norm(f\mathbf{W}g) = norm(f)\mathbf{W}norm(g)$, $norm(f\mathbf{U}g) = norm(f)\mathbf{U}norm(g)$
- $norm($**NOT NOT** $f) = norm(f)$
- $norm($**NOT** $(f$ **AND** $g)) = norm($**NOT** $f)$ **OR** $norm($**NOT** $g)$
- $norm($**NOT** $(f$ **OR** $g)) = norm($**NOT** $f)$ **AND** $norm($**NOT** $g)$
- $norm($**NOT** $\mathbf{X}f) = \mathbf{X}norm($**NOT** $f)$
- $norm($**NOT** $(f\mathbf{U}g)) = (norm($**NOT** $g)$ **AND** $norm(f))$ $\mathbf{W}(norm((\mathbf{NOT}\ f)\ \mathbf{AND}\ norm(\mathbf{NOT}\ g))$
- $norm($**NOT** $(f\mathbf{W}g)) = (norm($**NOT** $g)$ **AND** $norm(f))$ $\mathbf{U}(norm(\mathbf{NOT}\ f)\ \mathbf{AND}\ norm(\mathbf{NOT}\ g))$

Automata and $\mathcal{LTL}$ formulas

We wish to build an automaton which recognizes exactly the infinite sequences which check a formula (normalized) of $\mathcal{LTL}$. This automaton tries to establish a proof based on the propositions checked by the initial state of the sequence σ and on a formula to be checked by the suffix σ^1. Each state thus corresponds to a formula to check.

Let us suppose that we have to check the formula P **W** Q. According to equivalence f **W** $G \Leftrightarrow G$ **OR** $(F$ **AND X**$(F$ **W** $G))$,

- either in the initial state Q holds and σ^1 do not have a formula to check,
- or in the initial state P holds and σ^1 must check again P **W** Q.

We thus obtain the automat represented on the figure 2.

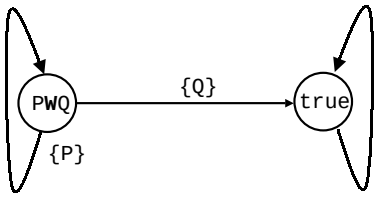


Figure 2: Automaton recognizing P **W** Q

A similar equivalence may be used for the âĂIJuntilâĂİ operator
f **U** $G \Leftrightarrow G$ **OR** $(F$ **AND X** $(F$ **U** $G))$.
However this automaton accepts the sequence where P holds indefinitely and Q never holds. The key point is that in the case of the operator **U**, one cannot indefinitely choose the second alternative of the **OR**. Let us note $\mathbf{X}^p$ an operator

who is a promise to check later an "until" formula by the first alternative of the **OR**. The automaton of the formula $P\mathbf{U}Q$ is depicted in figure 3. The semicolon present on the arc on the left separates the propositions to be checked and the promises to hold.

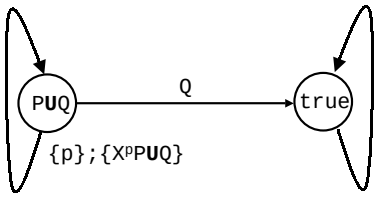


Figure 3: An automaton recognizing P **U** Q

The syntax and the semantics of the automata with promises is given below

Definition 7 *An automaton with promises* $A = \langle AP, Q, q_0, PROM, E\rangle$ *is defined by :*

- AP *a finite set of atomic propositions*
- Q *a finite set of states*
- $q_0 \in Q$ *the initial state*
- $Prom$ *a finite set of promises*
- E *a finite set of arcs such as for* $e \in E$
  - $in(E) \in Q$ *indicates the source of the arc*
  - $out(E) \in Q$ *indicates the target of the arc*
  - $label(E) \subset AP \cup \overline{AP}$ *indicates the propositions of the arc*
  - $prom(E) \subset Prom$ *indicates the promises associated with the arc*

Definition 8 *Let* $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ *be an infinite sequence of model* $\mathcal{KS}$. σ *is recognized by* $A = \langle AP, Q, q_0, Prom, E\rangle$ *if and only if there is a path* $(q_0, e_0, q_1, e_1, \ldots)$ *sucht that :*

- $\forall n, in(e_n) = q_n$, $out(e_n) = q_{n+1}$, $label(e_n) \subset \nu(s_n)$
- $\forall n, \forall pr \in prom(e_n), \exists m > n$ *such that* $pr \notin prom(e_m)$

A sequence which checks the first condition known as will be recognized by the way.

We consider the construction of an automaton equivalent to a formula f. As we saw on the examples, one transforms a formula into a disjunction of clauses where each clause is a conjunction of atomic propositions (and negations of

propositions) and formulas to be checked on the following under-sequence. We note $tr(F)$ the transformed formula where the formulas to be checked on the following under-sequence are replaced by propositions (noted like previously between hooks). The operator $\mathbf{X}^p$ is employed for the equivalence applied to the operator "until" : he indicates a promise to hold. Here the construction of this formula. It will be noted that the formula obtained is not presented syntactically in the form of a disjunction of conjunctive clauses. However this syntactic transformation is obtained by applying repeatedly equivalence f **AND** $(g$ **OR** $h) \Leftrightarrow (f$ **AND** $g)$ **OR** $(f$ **AND** $h)$. This transformation will be carried out during the construction of the automaton.
If $f = P$ **Then** $tr(f) = f$
If $f =$ **NOT** P **Then** $tr(f) = f$
If $f = h$ **OR** g **Then** $tr(f) = tr(h)$ **OR** $tr(g)$
If $f = h$ **AND** g **Then** $tr(f) = tr(h)$ **AND** $tr(g)$
If $f = \mathbf{X}g$ **Then** $tr(f) = [\mathbf{X}g]$
If $f = g\mathbf{U}h$ **Then** $tr(f) = tr(h)$ **OR** $(tr(g)$ **AND** $[\mathbf{X}^p g\mathbf{U}h])$
If $f = g\mathbf{W}h$ **Then** $tr(f) = tr(h)$ **OR** $(tr(g)$ **AND** $[\mathbf{X}g\mathbf{W}h])$

The construction of the automaton proceeds as follows :

- the initial state of the automaton is created and labelled with the formula to check.
- One applies the transformation to the formula described above. Each clause corresponds to an outgoing arc of the state. The target of the arc is a node labelled with the conjunction of formulas of the clause prefixed by a "next". The arc is labelled by the atomic propositions and the promises of the clause.
- One reiterates the process until there is no more new formula. What necessarily arrives since each formula is a conjunction of subformulas of the initial formula.
- For reasons of simplicity, one creates beforehand the state labelled by *true* which loopes on itself without proposition nor promise. This state is not necessarily reachable from the initial state.

One will find below a more formalized description of the algorithm. We will note $Aut(F)$, the automaton associated with f.

Create the state $(q_{true}, true)$
Create the arc e_{true} avec $in(e_{true}) = q_{true}$, $out(e_{true}) = q_{true}$,
$etiq(e_{true}) = \emptyset$, $prom(e_{true}) = \emptyset$
Create the state (q_0, f_0)
Insert (q_0, f_0) in $TODO$
While $TODO \neq \emptyset$ **do**
  Extract (q, f) from $TODO$
  Compute $tr(f)$
  Express $tr(f)$ in the form of a disjunction of conjunctive clauses

// $tr(f) =$ **OR**$_{c \in Cl}$
For each clause $c \in Cl$ **do**
// $c =$ **AND**$_{i \in I} P_i$ **AND**$_{j \in J}$ **NOT** Q_j **AND**$_{k \in K}[\mathbf{X} f_k]$ **AND**$_{l \in L}[\mathbf{X}^p g_l]$
If $f' =$ **AND**$_{k \in K} f_k$ **AND**$_{l \in L} g_l$ Ãľti quette un Ãľtat **Then**
Let (q', f') that state
Else
Create (q', f')
Insert (q', f') in $TODO$
End If
Create an arc e with $in(e) = q$, $out(e) = q'$
$etiq(e) = \{P_i\}_{i \in I} \cup \{\mathbf{NOT}\ Q_j\}_{j \in J}$, $prom(e) = \{\mathbf{X}^p g_l\}_{l \in L}$
End For
End While

Let $f = Q\mathbf{U}g$ with $g = (P$ **OR** $\mathbf{X}P)\mathbf{W}R$. Alors :
$tr(f) = tr(g)$ **OR** $(Q$ **AND** $[\mathbf{X}^p f])$
$tr(g) = R$ **OR** $((P$ **OR** $[\mathbf{X}P])$ **AND** $[Xg]) = R$ **OR** $(P$ **AND** $[Xg])$ **OR** $([\mathbf{X}P]$ **AND** $[\mathbf{X}g])$

Consequently, $tr(F)$ is the disjunction of 4 clauses :
- R which leads at the state labelled by *true* (more nothing to check)
- Q **AND** $[\mathbf{X}^p f]$ which loops on the initial state. It is noted that the infinite path which follows this arc is not accepted by the automaton because the promise $\mathbf{X}^p f$ is never held.
- P **AND** $[\mathbf{X}\ G]$ which leads at the state labelled by g.
- $[\mathbf{X}\ P]$ **AND** $[\mathbf{X}\ G]$ which leads at the state labelled by P **AND** g

Using $tr(G)$, the reader will check that the built automaton corresponds to the figure 4.

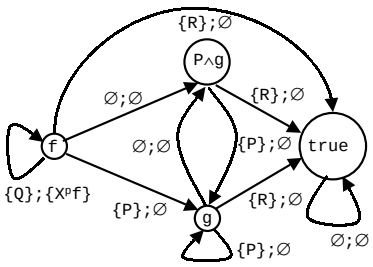


Figure 4: The automaton with promises of Q **U** $((P$ **OR** $\mathbf{X}P)$ **W** $R)$

Theorem 9 (Correction of the automaton) *Is f a formula of $\mathcal{LTL}$, then the sequences satisfying f are exactly those accepted by $Aut(F)$.*

Proof
Let cl be a clause of $tr(F)$. By definition, $cl \Rightarrow tr(F)$. We inductively define on

the size of f a set of subformulas g of f such as $cl \Rightarrow tr(G)$. We note this set $dev(Cl, F)$.
If $f = P$ **OR** $f =$ **NOT** P **OR** $f = \mathbf{X}g$ **Then**
  $dev(cl, f) = \{f\}$
Elsif $f = h$ **AND** g **Then**
  $dev(cl, f) = \{f\} \cup dev(cl, g) \cup dev(cl, h)$
Elsif $f = g$ **OR** h **Then**
  If $cl \Rightarrow tr(g)$ **Then**
    $dev(cl, f) = \{f\} \cup dev(cl, g)$
  Else $//cl \Rightarrow tr(h)$
    $dev(cl, f) = \{f\} \cup dev(cl, h)$
  End if
Elsif $f = g$ **U** h **Then**
  If $cl \Rightarrow tr(h)$ **Then**
    $dev(cl, f) = \{f\} \cup dev(cl, h)$
  Else $//cl \Rightarrow tr(g)$ **AND** $[\mathbf{X}^p g \mathbf{U} h]$
    $dev(cl, f) = \{f\} \cup dev(cl, g)$
  End if
Elsif $f = g$ **W** h **Then**
  If $cl \Rightarrow tr(h)$ **Then**
    $dev(cl, f) = \{f\} \cup dev(cl, h)$
  Else $//cl \Rightarrow tr(g)$ **AND** [**X** g **W** h]
    $dev(cl, f) = \{f\} \cup dev(cl, g)$
  End if
End if

Let $\sigma = (s_0, \ldots, s_i, \ldots)$ be a sequence accepted by a path of $Aut(F)$, $(q_0, e_0, \ldots, q_i, e_i, \ldots)$. Let us pose f_i the formula associated with q_i and cl_i the clause which produces the arc e_i. We show by recurrence on the size of the formula g that $\forall G \in Dev.(cl_i, f_i)\ \sigma^i \models g$.

If $g = P$ or $g =$ **NOT** P then g is a term of cl_i thus $g \in e_i$ what thus implies that $g \in \nu(s_i)\ \sigma^i \models g$.

If $g =$ **X** H then [**X** H] is a term of cl_i thus h is a constituent term of the conjunction f_{i+1}. By applying the assumption of recurrence, $\sigma^{i+1} \models h$ what implies $\sigma^i \models \mathbf{X}h$.

If $g = g_1$ **AND** g_2 then $\forall K, g_k \in Dev.(cl_i, f_i)$. By applying the assumption of recurrence, $\forall K\ \sigma^I \models g_k$ what implies $\sigma^i \models g$.

If $g = g_1$ **OR** g_2 then $\exists g_k \in Dev.(cl_i, f_i)$. By applying the assumption of recurrence, $\exists K\ \sigma^I \models g_k$ what implies $\sigma^i \models g$.

If $g = g_1$ **U** g_2 then

1. Either $cl_i \Rightarrow tr(g_2)$ and $g_2 \in Dev.(cl_i, f_i)$. By applying the assumption of

recurrence, $\sigma^I \models g_2$ what implies $\sigma^i \models g$.

2. Either $cl_i \Rightarrow tr(g_1)$ **AND** $[\mathbf{X}^p g_1 \mathbf{U} g_2]$. Then $g_1 \in Dev.(cl_i, f_i)$, $\mathbf{X}^p g \in PROM(e_i)$ and g is a term of the conjunction which constitutes f_{i+1}. By applying the assumption of recurrence, $\sigma^I \models g_1$. Since g is a term of the conjunction which constitutes f_{i+1}, we can apply the same reasoning to σ^{i+1}, σ^{i+2}, âĂę until the first alternative of the reasoning applies to σ^j with $j > i$. What arrives necessarily bus if not $\forall J \geq I, \mathbf{X}^p g \in PROM(e_j)$ contradicting the acceptance of the sequence by the path. Thus we have $\forall i \leq k < j$ $\sigma^k \models g_1$ and $\sigma^j \models g_2$, then $\sigma^i \models g$.

If $g = g_1$ $\mathbf{W}$ g_2 then

1. Either $cl_i \Rightarrow tr(g_2)$ and $g_2 \in dev(cl_i, f_i)$. By applying the assumption of recurrence, $\sigma^i \models g_2$ and $\sigma^i \models g$.
2. Either $cl_i \Rightarrow tr(g_1)$ **AND** $[\mathbf{X} g_1 \mathbf{W} g_2]$. then $g_1 \in dev(cl_i, f_i)$ and g is a term of the conjunction which constitutes f_{i+1}. By applying the assumption of recurrence, $\sigma^i \models g_1$. Since g is a term of the conjunction which constitutes f_{i+1}, the same reasoning applies to σ^{i+1}, σ^{i+2},... and :
   - Either the first alternative of the reasoning applies to a sequence σ^j with $j > i$. In that case, $\forall i \leq k < j$ $\sigma^k \models g_1$ and $\sigma^j \models g_2$. Consequently, $\sigma^i \models g$.
   - Either $\forall j \geq i, \sigma^i \models g_1$ and consequently $\sigma^i \models g$.

Since $f = f_0$, $\sigma \models f$.

Let us suppose now that $\sigma \models f$. We built now a path in $Aut(f)$ which recognizes σ. First, we recursively define a clause of $tr(f)$ depending from σ :
$cl(f, \sigma) = \mathbf{AND}_{i \in I} P_i$ $\mathbf{AND}_{j \in J}$ **NOT** Q_j $\mathbf{AND}_{k \in K}[\mathbf{X} f_k]$ $\mathbf{AND}_{l \in L}[\mathbf{X}^p g_l]$
such that :
$\sigma \models \mathbf{AND}_{i \in I} P_i$ $\mathbf{AND}_{j \in J}$ **NOT** Q_j $\mathbf{AND}_{k \in K} \mathbf{X} f_k$ $\mathbf{AND}_{l \in L} \mathbf{X} g_l$.
Its definition follows :

If $f = P$ **Then** $cl(f, \sigma) = f$
Elsif $f =$ **NOT** P **Then** $cl(f, \sigma) = f$
Elsif $f = \mathbf{X} g$ **Then** $cl(f, \sigma) = [\mathbf{X} g]$
Elsif $f = g$ **AND** h **Then** $cl(f, \sigma) = cl(g, \sigma)$ **AN D** $cl(h, \sigma)$
Elsif $f = g$ **OR** h **Then**
  If $\sigma \models g$ **Then**
    $cl(f, \sigma) = cl(g, \sigma)$
  Else // $\sigma \models h$
    $cl(f, \sigma) = cl(h, \sigma)$
  End if
Elsif $f = g \mathbf{U} h$ **Then**
  If $\sigma \models h$ **Then**
    $cl(f, \sigma) = cl(h, \sigma)$
  Else // $\sigma \models g$ **AND** $\mathbf{X} f$
    $cl(f, \sigma) = cl(g, \sigma)$ **AND** $[\mathbf{X}^p f]$

End if
Elsif $f = g\mathbf{W}h$ **Then**
If $\sigma \models h$ **Then**
$cl(f,\sigma) = cl(h,\sigma))$
Else // $\sigma \models g$ **AND** $\mathbf{X}f$
$cl(f,\sigma) = cl(g,\sigma)$ **AND** $[\mathbf{X}f]$
End if
End if

Let e be the arc associated with $cl(f,\sigma)$, $q_1 = out(e)$ and f_1 the formula associated with q_1. By construction, $prop(e) \subset \nu(s_0)$ and $\sigma \models \mathbf{X}f_1$. Then $\sigma^1 \models f_1$ and it is possible to iterate the construction leading a path recognizing σ. Let us suppose the existing of a promise $\mathbf{X}^p g\mathbf{U}h$ occuring on the path at the rank i. By construction of the clause, we have $\sigma^i \models g\mathbf{U}h$ but in that case, there exists a rank $j \geq k$ such that $\sigma^j \models h$ and consequently the clause associated with σ^j does not does not comprise promises. Finally, this path accepts σ. $\diamondsuit$

$\mathcal{LTL}$ formulas may be represented by others models of automata. Here, we essentially have followed the approach described in [COU 99]. The most widespread model is certainly that of Büchi automata [BUC 62]. Their syntax and semantics is given below, the interested reader may refer to [VAR 96] for a detailled study between temporal logic and automatas.

Definition 10 *A Büchi automaton* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F\rangle$ *is defined as follows :*

- AP *a finite set of atomic propositions*
- Q *a finite set of states such that for* $q \in Q$, $etiq(q) \subset AP \cup \overline{AP}$ *is the set of atomic propositions which holds in that statee*
- $Q_0 \subset Q$ *the subset of initial states*
- $\rightarrow$ *is the transition relation* $\subset Q \times Q$
- $F \subset Q$ *the subset of succes states*

Definition 11 *Let* $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ *be an infinite sequence of the model* $\mathcal{KS}$. σ *is recognized by* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F\rangle$ *if and only if there exists a path* $(q_0, q_1, \ldots)$ *with* $q_0 \in Q_0$ *sucht that :*

- $\forall n, q_n \rightarrow q_{n+1}$ *and* $etiq(q_n) \subset \nu(s_n)$
- $\exists f \in F$ *such that* $\forall n\ \exists m > n\ q_m = f$

It will be noted that the propossitions relate on the states and either to the transitions, that one has a set of initial states and that the condition of acceptance is defined by a set of states of success whose at least state must be reached an infinity of time by the path.

The expressiveness of Büchi's automata and automata with promises is identical. It is important to note that $\mathcal{LTL}$ has an expressiveness more restricted than these automatas models [WOL 83]. Another language of formulas (much less intuitive), the linear μ-calcul has as for him an expressiveness equivalent to these models [DAM 92]. We informally explain the translation of an automaton with promises out of Büchi's automaton :

- Let us suppose that we have n promises. For each arc e of the automaton with promises, one builds $n+1$ states of the Büchi's automaton $\{(q_e, I)\}_{I \in 1...n+1}$ with $etiq((q_e, I)) = etiq(E)$.
- the initial states of the automaton are the $(q_e, 1)$ such as $in(E) = q_0$.
- For $i \leq n$, there is an arc of (q_e, I) towards $(q_{e'}, I)$ if $out(E) = in(e')$ and if Pr_i belongs to $prom(e')$.
- For $i \leq n$, there is an arc of (q_e, I) towards $(q_{e'}, i+1)$ if $out(E) = in(e')$ and if Pr_i does not belong to $prom(e')$.
- There is an arc of $(q_e, n+1)$ towards $(q_{e'}, 1)$ if $out(E) = in(e')$.
- the states of success are the states $\{(q_e, n+1)\}$.

The transformation of the arcs into states is usual and does not require particular comments. When during the recognition of a sequence, we find a state (q_e, I) with $i \leq n$, we wait until the promise Pr_i is held. If it is not it in the next state, one passes in a of the same state index i if not one passes in a subscripted state by $i+1$.

Arrived in a state of index $n+1$, all the promises were held at least once and the examination of the promises then is started again. Thus if the promises are indefinitely held, one passes an infinity of time by the states of index $n+1$ whereas in the contrary case one "stagnates" in a subset of states of index $i \leq n$. We leave to the reader the care to find a transformation of an Büchi's automaton into an automaton with promises. The figure 5 illustrates this construction. To simplify, one removed the nonaccessible states. The states "white" correspond to index 1, gray states the "dark" correspond to index 2 and gray states the "clearly" correspond to index 3. The initial states are depicting by an entering arc. The fatty arcs indicate transitions between of the same states index, whereas the fine arcs are associated changes of index.

Existence of a sequence accepted by a Büchi automaton

The existence of an infinite sequence σ of a model $\mathcal{KS}$ accepted by a Büchi automaton is established using a standard construction the so-called synchronized product.

Definition 12 *Let $\mathcal{KS}$ be a model and $\mathcal{B}$ a Büchi automaton then $\mathcal{KS} \times \mathcal{B} = (AP', S', \rightarrow', \nu')$ is defined as follows :*

- *$AP' = AP$ is a finite set of atomic propositions,*

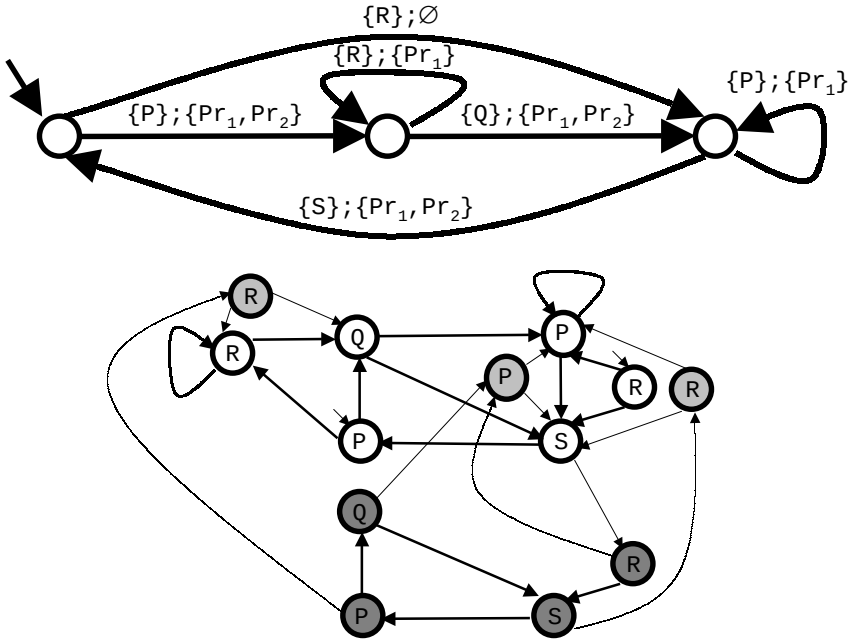


Figure 5: Transformation of an automaton with promises to a Büchi automaton

- $S' = \{(s,q) | s \in S, q \in Q, etiq(q) \subset \nu(s)\}$
- $(s,q) \rightarrow' (s',q') \Leftrightarrow s \rightarrow s'$ *and* $q \rightarrow q'$
- $\nu'(s,q) = \nu(s)$

In an obvious way, the synchronized product generates the infinite sequences whose first component (an infinite sequence of $\mathcal{KS}$) is recognized by the second component (an infinite path of Q). It remains us to be checked if the synchronized product contains an infinite sequence begin with (s_0, q_0) with $q_0 \in Q_0$ and whose second component contains an infinity of occurrences of states of F. It is the object of the following theorem. A strongly related component (s.c.c.) of a graph is elementary if the subgraph associated with this s.c.c is a single top without loop (in other words, it is not possible to build an infinite path in this s.c.c).

Theorem 13 *Let $\mathcal{KS}$ be a finite model, s_0 a state of $\mathcal{KS}$ and $\mathcal{B}$ a Büchi's automaton then :*
$\exists \sigma = (s_0, s_1, \ldots)$ a sequence $\mathcal{KS}$ accepted by $B \Leftrightarrow \exists \mathcal{C}$ a nonelementary s.c.c of $\mathcal{KS} \times B$ accessible from one $(s_0, q_0) \in S'$ with $q_0 \in Q_0$ containing a state (s, f) with $f \in F$

Proof
Let $\sigma = (s_0, s_1, \ldots)$ be a sequence of $\mathcal{KS}$ accepted by $(q_0, q_1, \ldots)$ a path of $\mathcal{B}$ then by construction $((s_0, q_0), (s_1, q_1), \ldots)$ is a sequence of $\mathcal{KS} \times \mathcal{B}$ which meets an infinity of time states of the form (s, f) with $f \in F$. Since $\mathcal{KS} \times \mathcal{B}$ is finite, one of these states (noted (s^*, f^*)) is reached an infinity of time by the sequence. Since from (s^*, f^*) one again reaches this state by a non null sequence, the s.c.c containing (s^*, f^*) is non elementary. Since the first state of the sequence is (s_0, q_0) this s.c.c is accessible from (s_0, q_0).

If the left member of equivalence is checked, then there is a finite sequence σ_1 of (s_0, q_0) towards (S, F) and one non null finite sequence σ_2 of (S, F) worms itself. Consequently, $\sigma = \sigma_1.\sigma_2^\infty$ is an infinite sequence whose second component (a path in $\mathcal{B}$) accepts the first (a sequence of $\mathcal{KS}$). $\diamondsuit$

This result provides us an effective means of verification : once built the synchronized product, one calculates the s.c.c by means of the algorithm of Tarjan [AHO 74] and one examines them. The size of the synchronized product is proportional to the sizes of the model and the formula. The algorithm, as for him, operates in a polynomial time according to the size of the synchronized product.

However this effectiveness is only apparent. On the one hand, the size of the model of execution is very large relative with the size of the model of specification (e.g size of the graph of accessibility versus size of the Petri net). In addition, the size of the automaton can be an exponential function of the size of the formula. This last point is not also critical because the formulas are generally of very reduced size. Also to reduce these problems of complexity, different technical were proposed. Upstream, one seeks to check the formula on a smaller model of execution but equivalent (see the following chapters). Downstream, one seeks to check a formula without completely developing the product synchronized by methods "on the fly" [GOD 93, GER 95] or to reduce the size of the representation by structures of data of the type BDD (diagrams of binary decision) [BRY 86].

3.3. *Temporal logic and Petri nets*

The specification of formulas of propositional temporal logic of Petri net implies the definition of atomic properties. Since we evaluate the formulas of temporal logic on the graph of accessibility, a state of the model is an accessible marking. Also, for any boolean expression whose field is the set of markings of the net is appropriate. In practice, the expressions used are evaluated easily. One will note p for the marking of p in the current state. Here some examples of frequent formulas.

- Two places p_1, p_2 are mutually exclusives : $\mathbf{AG}\ (p_1 \cdot p_2 = 0)$
- For any reachable marking, a place p will be inevitably marked : $\mathbf{AG}\ \mathbf{AF}(p > 0)$
- For any reachable marking, one can always mark a place p : $\mathbf{AG}\ \mathbf{EF}\ (p > 0)$
- During any sequence of execution, a place p is indefinitely marked and unmarked $\mathbf{AG}\ (\mathbf{F}\ (p > 0)\ \mathbf{AND}\ \mathbf{F}\ (p = 0))$
- A transition t is live (always firable in the future of any state) : $\mathbf{AG}\ \mathbf{EF}\ \ \mathbf{AND}_{p \in P}\ (p \geq Pré(p, y))$

Like illustrates it the last example, it is possible to reason on the franchissability of a transition. However it is not the case of crossing itself because it would require to evaluate the evolution of the marking of the places between two successive states. Also one extends the language $\mathcal{CTL}^*$by considering the operator $\mathbf{X}_{\{E\}}$ whose semantics is defined by :
$\sigma \models \mathbf{X}_{\{e\}} f$ if and only if $\mathcal{KS}, \sigma^1 \models f$ and the first transition from σ is labelled by e

In this paragraph, one considers that a transition from Petri net is never labelled by the empty word. The methods of checking described above extend in an immediate way to this new logical which is at the same time state and event-based. Let us suppose that the labelling of a net is the identity. We can now express the fact that a transition t is indefinitely crossed in all sequence : $\mathbf{AGFX}_{\{t\}} true$.

When we study the decidability of the checking of formulas of temporal logic on Petri net, we will distinguish the following cases :

- state-based logic $\mathcal{CTL}^*$(and its fragments) by prohibiting these new operators.
- event-based logic $\mathcal{CTL}^*$(and its fragments) if the only atomic propositions are $true$ and $false$.

The semantics of temporal logic is based on the infinite sequences but the designer also wishes to reason on the finite sequences. For example, one wishes to know if a place p is always marked in a dead marking. The following formula $\mathbf{AG}(\ \mathbf{AND}_{t \in T}\ \mathbf{NOT}\ \mathbf{X}_{\{t\}} true \Rightarrow p > 0)$ which seems to be appropriate is incorrect because it is actually a tautology. Indeed, an infinite sequence never satisfies $\mathbf{AND}_{t \in T} \mathbf{NOTX}_{\{t\}} true$.

To take into account these needs for checking, it is necessary to distinguish the type of studied sequence and to introduce a semantics of adequate satisfaction. Since we treat the sequences, we consider that the path formulas are represented by an automaton such as the arcs of this automaton are labelled by labels of transition from Petri net. In order not to weigh down the presentation by an enumeration of all the possible cases, we limit ourselves to an event-based linear logic defined by means of labelled Büchi's automaton.

Definition 14 *A labelled Büchi's automaton* $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}, F\rangle$
- Σ *an finite alphabet*
- Q *a finite set of states*
- $Q_0 \subset Q$ *the subset of the initial states*
- $\stackrel{has}{\longrightarrow}$ *is a binary relation* $\subset S \times S$
- $F \subset Q$ *a subset of success states*

Definition 15 *An infinite sequence* $\sigma = (t_1, t_2, \ldots, t_n, \ldots)$ *of a Petri net* R, σ *is accepted by* $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}, F\rangle$ *if and only if there is a path* $(q_0, q_1, \ldots)$ *with* $q_0 \in Q_0$ *such as :*
- $\forall N, q_n \stackrel{L(t_{n+1})}{\longrightarrow} q_{n+1}$
- $\exists F \in F$ *s.t.* $\forall N\ \exists m > n\ q_m = f$

The other types of sequence which interest the designer are the finite sequences and the finite maximal sequences (i.e which end in a dead marking). One then seeks a path in the automaton which is completed by a state of success. A second manner of tackling the problem of the finite maximal sequences in the case of bounded Petri Net consist with adding a loop to all markings died, labelled by a special action. Thus any maximal sequence of this new graph is infinite and those which are prolonged artificially recognize by the occurrence of the special action.

Definition 16 *A finite sequence (possibly maximal)* $\sigma = (t_1, t_2, \ldots, t_f)$ *of a Petri Net* R, σ *is recognized by* $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}, F\rangle$ *if and only if there is a path* $(q_0, q_1, \ldots, q_f)$ *with* $q_0 \in Q_0$ *and* $q_f \in F$ *such that :* $\forall n < f, q_n \stackrel{l(t_{n+1})}{\longrightarrow} q_{n+1}$

To authorize the labelling of a transition by the empty word (i.e a non observable transition) largely complicates the semantics of satisfaction and introduced the problem of the divergence. A divergent sequence is an infinite sequence of which a suffix is made up exclusively of non observable transitions.This problem will be mentioned in the context of the behavioral approach.

4. Behavioral Approach

Many relations of equivalence were used or specifically proposed for the comparison and the analysis of concurrent systems since the equivalence of traces or languages [AHO 74] with the observational equivalence [MIL 89] while passing by the models of refusal and the equivalences of test [LED 90, BRI 88]. See [DE 87, ARN 92, OH 86, GLA 90] for a panorama of the existing equivalences.

This explosion is explained on the one hand, by the difficulty in formally defining a universal semantics of concurrent systems [ARN 92] and by the variety of the specific properties of the studied systems or the points of view which one can adopt to lead to them analyze : verification or test. In this section, we will limit ourselves to the aspect verification.

Contrary to the logical approach, the behavioral approach privilegiates information associated with the action labels and generally forgets information associated with the states. The structure taken into account by the behavioral analysis is a system of transitions labelled (cf definition 3).

Before going front in formalization, we will try to illustrate various points of view likely to be taken into account. The selected example is the simplified operation of a coffee machine : the consumer introduces a coin into the coiner, it chooses then his drink while pressing on the associated button.

The transition systems below, $\langle D, 0\rangle, \langle D', 0'\rangle$ and $\langle D'', 0''\rangle$, represent each one a behavior "similar" to the vending machine which we have just described. The behavioral approach through various equivalences of behaviors which were proposed in the literature will enable us to formalize various concepts of "similarity".

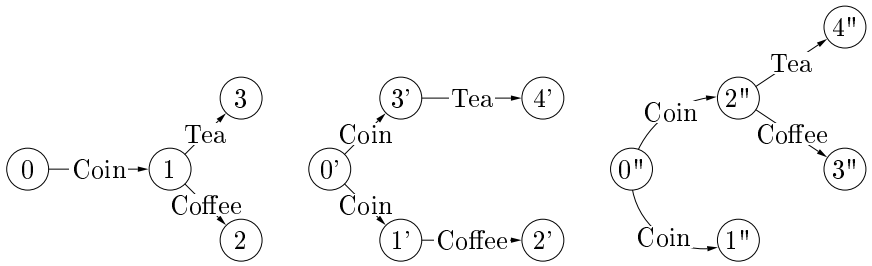


Figure 6: Three vending machines : $\langle D, 0\rangle, \langle D', 0'\rangle$ and $\langle D'', 0''\rangle$

According what was presented in chapter 3 of volume on the Petri nets [HV 01], one can associate any ILTS a language.

Definition 17 *Language associated with a ILTS*

Let $\langle \mathcal{LTS}, s_0\rangle$ *be a ILTS with* $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$,

$L(\langle \mathcal{LTS}, s_0\rangle) =_{Def} \{\sigma \in \Sigma^* \ : \exists \in S \text{ such that } \ s_O \stackrel{\sigma}{\rightarrow}\}$

Contrary to finite states automatas, the ILTS can comprise an infinity of states, they comprise only one initial state and do not introduce the concept of final state [AHO 74]. Any state of the ILTS is thus regarded as a final state

and the language recognized by the ILTS is closed by prefix : any prefix of a recognized word itself is recognized.

Definition 18 *Equivalence langage*

The concept of language previously introduced enables us to define a first concept of equivalence between two transitions systems based on the equality from their respective languages.

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *be two transitions systems,* s_0 *and* s'_0 *their respective initial states :*

$\langle \mathcal{LTS}, s_0\rangle \equiv \langle \mathcal{LTS}', s'_0\rangle$ *iff* $L(\langle \mathcal{LTS}, s_0\rangle) = L(\langle \mathcal{LTS}', s'_0\rangle)$

Langage : A first comparison criterion of these distributers is provided to us by the study of their language.

Here $L(\langle D, 0\rangle) = L(\langle D, 0'\rangle) = L(\langle D'', 0''\rangle) = \{\epsilon, \text{Part, Coin.Coffee, Coin.Tea}\}$ and for this criterion these three LTS are equivalents. In particular, from the point of view of the owner of the vending machine, each one of them offer a drink only if this one has been payed.

Maximal traces : The preceding criterion disregards possibility of blocking, it confuses these 3 LTS whereas they present different deadlocks. The concept of maximal trace allow to take into account this aspect. The LTS are always considered as acceptors of language, but now maximal sequences are the only one recognized : infinite sequences or sequences leading on states without successor.

Definition 19 *Maximal Traces*

For $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and a ILTS* $\langle \mathcal{LTS}, s_0\rangle$*, one associates* $L_{max}(\langle \mathcal{LTS}, s_0\rangle)$ *the set of its maximal traces defined as follows :*

$$\begin{aligned} L_{Max}(\langle \mathcal{LTS}, s_0\rangle) \quad &=_{Def} (L(\langle \mathcal{LTS}, s_0\rangle) \cap \Sigma^\infty) \\ &\cup \ \{\sigma \in L(\langle \mathcal{LTS}, s_0\rangle) \ : \exists s' \in S \text{ such that } \quad s \stackrel{\sigma}{\rightarrow} s' \text{ and } s' \not\rightarrow\} \end{aligned}$$

Definition 20 *Maximal traces equivalence*

Provided with this concept of maximal trace, we define the relation of associated equivalence as follows :

Let $\langle \mathcal{LTS}, s_0\rangle$ *and* $\langle \mathcal{LTS}', s'_0\rangle$ *be two transitions systems,* s_0 *and* s'_0 *their respective initial states :*

$\langle \mathcal{LTS}, s_0\rangle \equiv_{Max} \langle \mathcal{LTS}', s'_0\rangle$ *iff* $L_{Max}(\langle \mathcal{LTS}, s_0\rangle) = L_{Max}(\langle \mathcal{LTS}', s'_0\rangle)$

Remark Language and Maximal Traces

The definition 19 is purely denotational, from an operational point of view the concept of "maximal traces" can be expressed starting from the concept of language in "supplementing" the ILTS (or the automaton) by adding : a state $\perp$ with the set of nodes ($\perp \notin S$), a label $fail$ with the alphabet Σ ($fail \notin \Sigma$) and, finally, by connecting any deadlock to the state $\perp$ by a transition labelled by $fail$.

Property 21 *Language and Maximal Traces*

For $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$, *one defines* $Max(\mathcal{LTS})$ *as follows :*

$$Max(\mathcal{LTS}) =_{Def} \mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$$

$$\textit{where :} \begin{cases} \Sigma' = \Sigma \cup \{fail\}, \\ S' = S \cup \{\perp\}, \\ \{\stackrel{a'}{\rightarrow}{}'_{a\in\Sigma'}\} =_{Def} \{\stackrel{a}{\rightarrow}_{a\in\Sigma}\} \cup \{ \quad s \stackrel{fail}{\rightarrow} \perp \ : s \in S \textit{ such that } \stackrel{s}{\not\rightarrow}\} \end{cases}$$

$$L_{max}(\langle \mathcal{LTS}, s_0\rangle) = L_{max}(\langle \mathcal{LTS}', s'_0\rangle)\sigma$$
$$L(\langle max(\mathcal{LTS}), s_0\rangle) = L(\langle max(\mathcal{LTS}'), s'_0\rangle)$$

Now we have, $L_{Max}(\langle D, 0\rangle) = L_{Max}(\langle D', 0'\rangle) = \{$Coin.Coffee, Coin.Tea$\}$ but $L_{Max}(\langle D'', 0''\rangle) = \{$Coin, Coin.Coffee, Coin.Tea$\}$.

According to this new criterion, only D and D' remain equivalent. If one takes into account now the point of view of the customer, it is indeed important to isolate the distributor D'' which can to accept a coin without delivering drink. Always according to the point of view of the customer, not to be able to distinguish D and D' is not acceptable : D leaves the choice of drink to the customer while D' chooses drink in its place.

Refual & Acceptance : The equivalence of maximal traces takes into account "total blockings", the equivalences based on the concept of refusal or of acceptance, allow to take into account the concept of partial blockings and in particular the possibility "of refusing" to carry out an action. Thus, one can consider, in addition to the allowed sequences, the possibility of refusing or of accepting an action.

Definition 22 *Basic elements of refusal semantics [GLA 90, LED 90]*

Let $\langle \mathcal{LTS}, s_0\rangle$ *be a ILTS, for* $s \in S, \sigma \in \star\Sigma$ *and* $A \subset \Sigma$, *we note :*

1. $s \ ref \ A \Leftrightarrow_{Def} \forall a \in A, s \stackrel{a}{\not\rightarrow}$
2. $s \models after\ \sigma \Leftrightarrow_{Def} \{s' \in S \ : \ s \stackrel{\sigma}{\rightarrow} s'\}$
3. $s \models after\ \sigma \ ref \ A \Leftrightarrow_{Def} \exists s' \in$ "$s \ after\ \sigma$" *such that* $s' \ ref \ A$

4. $\mathcal{LTS} \models \ after\ \sigma\ ref\ A \Leftrightarrow_{Def} s_0 \models \ after\ \sigma\ ref\ A$

(1) allows to define partial blockings partial through the refusal set which one can associate a node. (2) denotes the subset of the nodes accessible starting from node s via the sequence σ (3) stipulates that "starting from node s, it is possible, via the sequence σ, to reach a node which will refuse all the actions of A

Definition 23 *Relation of Conformance [BRI 88, LED 90] Let* $\langle \mathcal{LTS}, s_0 \rangle$ *and* $\langle \mathcal{LTS}', s'_0 \rangle$ *be two ILTS and L the unionset of their respective alphabets* $(L = \Sigma \cup \Sigma')$

$$\mathcal{LTS}\ \underline{conf}\ \mathcal{LTS}' \Leftrightarrow_{Def} \left\{ \begin{array}{l} \forall \sigma \in L(\langle \mathcal{LTS}, s_0 \rangle), \forall A \subset L : \\ \quad If\ \mathcal{LTS}\ after\ \sigma\ ref\ A\ then\ \mathcal{LTS}'\ after\ \sigma\ ref\ A \end{array} \right.$$

In an informal way, an implementation $\mathcal{LTS}$ conforms to a specification $\mathcal{LTS}'$ if for any sequence σ, if the implementation can evolve by σ then the set of actions A which it can refuse constitutes a subset of those which the specification can refuse after σ [DRI 92].

Definition 24 *Testing equivalence [BRI 88]*

$$\mathcal{LTS}\ \underline{te}\ \mathcal{LTS}' \Leftrightarrow_{Def} \left\{ \begin{array}{l} L(\langle \mathcal{LTS}, s_0 \rangle) = L(\langle \mathcal{LTS}', s'_0 \rangle) \\ \mathcal{LTS}\ \underline{conf}\ \mathcal{LTS}'\ and\ \mathcal{LTS}'\ \underline{conf}\ \mathcal{LTS} \end{array} \right.$$

For this last point of view, which melts the semantics based on refusal, the LTS D and Of are not "testing-equivalent" (not D te Of). Indeed D' can refuse Tea or Coffee actions after having carried out the Coin action while D after Coin will always make it possible to obtain Tea or Coffee. By taking again the elements of terminology of the definition 22, one obtains for example :

$0'\ after$ Coin ref {Tea, Coin} et $0'\ after$ Coin ref {Coffee, Coin} while the only action refused from 0 is Coin, i.e., $0\ after$ Coin ref {Coin}.

We will not develop more before these semantics (failure semantics), but we invite the interested reader to refer to [ARN 92] where chapter 8 is devoted to various equivalences of traces.

4.1. *Relations of Bisimulation*

The three equivalences which we have just evoked adopt a "linear" point of view and focusses on the sequences of executions of the LTS and disregard its tree structure. To illustrate our matter, now let us consider a coffee machine

where the only drink available is the coffee sweetened : the customer introduces a coin and must obtain the coffee then sugar.

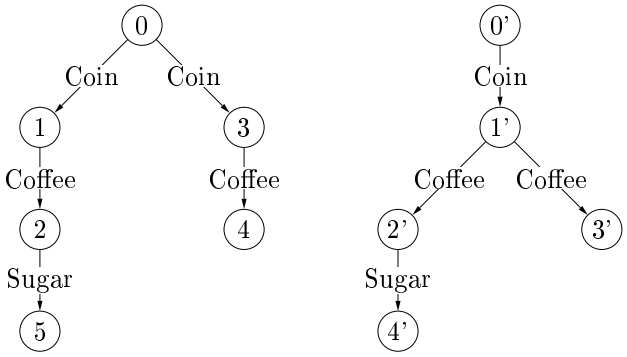


Figure 7: Two coffee machines : $\langle M, 0\rangle$ and $\langle Me, 0'\rangle$

The machines represented figure 7 are indistinguishable for the semantic based on refusal or for testing equivalences [BRI 88]. The two machines can refuse sugar after having delivered drink. For $s \in \{0, 0'\}$ we have :
$s\ after$ Coin ref {Coin, Sugar} and $s\ after$ CoinCoffee ref {Coin, Coffee, Sugar}

From the point of view of the customer these two machines are thus as much imperfect. A customer who can test, as a long time as he wants it these two machines, is unable to distinguish them. For each one of those, certain experiments will result in obtaining a sweetened coffee and others with a coffee without sugar.

From the point of view of the analysis and in particular if one seeks to understand why one cannot guarantee to the customer whom it will obtain a sweetened drink it is however interesting of distinguish them. in the first case, the absence of sugar will be consecutive of the non-determinism associated with the Coin action while in the second it will result from the non-determinism associated with the Coffee action. The concept of bisimulation which takes into account the tree structure of the LTS, and not only its linear structure, will enable us to distinguish these two machines.

The concept of Bisimulation, introduced by Park [PARK 81] is at the base many relations of equivalences used for the checking of communicating systems.

Definition 25 *Relation of Bisimulation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ be a LTS, B a binary relation ($B \subset S \times S$) is a relation of bisimulation if it verifies :

$\forall (p,p') \in B$ *and* $\forall t \in \Sigma$:

$$[\forall q \in S \ \textit{If} \quad p \xrightarrow{t} q \ \textit{then} \ \exists q' \in S \ : \quad p' \xrightarrow{t} q' \ \textit{and} \ (q,q') \in B]$$
$$\textit{and} \quad [\forall q' \in S \ \textit{If} \quad p' \xrightarrow{t} q' \ \textit{then} \ \exists q \in S \ : \quad p \xrightarrow{t} q \quad \textit{and} \ (q',q) \in B]$$

Two states $s_1, s_2 \in S$ are in bisimulation if there is a relation of bisimulation B such as $(s_1, s_2) \in B$.

Definition 26 *Bisimulation between transitions systems*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ and $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ be two LTS such as $S \cap = \emptyset$ and for which we let $\mathcal{S} = S \cup S'$

A binary relation B, ($B \subset \mathcal{S} \times \mathcal{S}$) is a relation of bisimulation between $\mathcal{LTS}$ and $\mathcal{LTS}'$ if it verifies :

$\forall (p,p') \in B$ *and* $\forall t \in \Sigma \cup \Sigma'$:

$$[\forall q \in \mathcal{S} \ \textit{If} \quad p \xrightarrow{t} q \ \textit{then} \ \exists q' \in \mathcal{S} \ : \quad p' \xrightarrow{t} q' \ \textit{and} \ (q,q') \in B]$$
$$\textit{and} \quad [\forall q' \in \mathcal{S} \ \textit{If} \quad p' \xrightarrow{t} q' \ \textit{then} \ \exists q \in \mathcal{S} \ : \quad p \xrightarrow{t} q \quad \textit{and} \ (q',q) \in B]$$

Definition 27 *Bisimilar transitions systems*

The preceding definition extends in a canonical way to initialized transitions systems while posing that two ILTS $\langle \mathcal{LTS}, s_0\rangle$ and $\langle \mathcal{LTS}', s'_0\rangle$ are in bisimulation (or are bisimilar) if a bisimulation relation connects their initial respective states. i.e.

$\langle \mathcal{LTS}, s_0\rangle$ and $\langle \mathcal{LTS}', s'_0\rangle$ are in bisimulation if $\exists$ a bisimulation $B \subset \mathcal{S} \times \mathcal{S}$ between $\mathcal{LTS}$ and $\mathcal{LTS}'$ such as $(s_0, s'_0) \in B$

Example 1

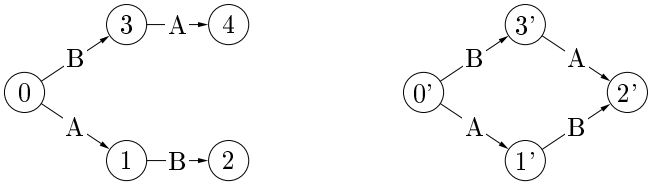


Figure 8: Two bisimilar LTS : $\langle E, 0\rangle, \langle E', 0'\rangle$

$\langle E, 0\rangle, \langle E', 0'\rangle$, represented 8, are in bisimulation by relation $B = \{(0,0')(1,1'),(2,2')(3,3')(4,2')\}$.

$\langle D, 0\rangle, \langle D', 0'\rangle$ and $\langle D'', 0''\rangle$, associated the drink distributers represented 6 are not bisimilar. Let us show for example that $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ are not in bisimulation. Let us proceed by the absurb and suppose the existence of a bisimulation B between D and D' with $(0, 0') \in B$. Like $0 \stackrel{\text{Coin}}{\rightarrow} 1$, one must have $(1, 1') \in B$ or $(1, 3') \in B$. $(1, 1') \in B$ is impossible because $1 \stackrel{\text{Tea}}{\rightarrow}$ and $1' \stackrel{\text{Tea}}{\not\rightarrow}$. In the same way $(1, 3') \in B$ involves a contradiction because $1 \stackrel{\text{Coffee}}{\rightarrow}$ and $3' \stackrel{\text{Coffee}}{\not\rightarrow}$.

The ILTS $\langle M, 0\rangle, \langle M', 0'\rangle$, associated the coffee machines represented 7, are not bisimilar. Even if these ILTS are small, it quickly becomes difficult to show "with the hand", starting from the denotational definition of the bisimulation (cf def 25), the existence or the nonexistence of a bisimulation.
the section 4.1.1 gives an algorithm making it possible to decide bisimulation which will enable us to rule on these two systems.
the section 4.3.4, by introducing the logic of Hennesy-Milner, will give us the elements of language making it possible to distinguish without ambiguity these two systems and to show that they are not bisimilar.

Property 28 *Properties of the bisimulations : [ARN 92]*

- *The converse relation of a bisimulation is also a bisimulation.*
- *the composition of 2 bisimulations is a bisimulation.*
- *the union of 2 bisimulations is a bisimulation.*

The properties above make it possible to define a specific relation of bisimulation, the strong equivalence which is the largest bisimulation.

Definition 29 *Strong Equivalence Strong equivalence, noted $\sim$, is defined by*

$p \sim q \Leftrightarrow_{Def}$ *there exists a bisimulation*B *such as*$(p, q) \in B$

$\sim$ *is reflexive because the identity is a bisimulation. Symmetry and transitivity come respectively owing to the fact that the set of the bisimulations is stable respectively by inversion and composition.*

4.1.1. *Algorithm of Decision of Bisimulation*

This section shows how to build, if it exists, a relation of bisimulation starting from any system of transition finitary. The property 32 melts the algorithm of decision. We present also some elementary concepts and properties which make it possible "to compact" computation.

Definition 30 *Finitary LTS*

A LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *is called finitary, or in an equivalent way, has a "finite image" relation of accessibility :*

$\forall S \in S$ *and* $\forall a \in \Sigma$, *the set* $\{Q \in S$ *such that* $S \stackrel{a}{\rightarrow} Q\}$ *is finite.*

Definition 31 $\sim_N$ *equivalences*

For a LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$, *one considers the following sequence of relations indexed by* i, *noted* $\sim_I$: $\forall p, q \in S$

- $p \sim_0 q$
- $p \sim_{n+1} q$ *ssi* $\forall a \in \Sigma$

$$\begin{array}{lll} \forall p' \in S & p \stackrel{a}{\rightarrow} p' \Rightarrow \exists q' \in Q & : \quad q \stackrel{a}{\rightarrow} q' \text{ such that } p \sim_n p' \\ \forall q' \in S & q \stackrel{a}{\rightarrow} q' \Rightarrow \exists p' \in Q & : \quad p \stackrel{a}{\rightarrow} p' \text{ such that } q \sim_n q' \end{array}$$

Intuitively, one tests the $\sim_n$-equivalence between two systems as follows. For each system, one builds the tree of the sequences lower length or equal with n (obtained by regarding various occurrences of the same state as different states). Then it is checked that these two trees are bisimilar. The following property specifies the relations between $\sim_n$-equivalence and bisimulation.

Property 32 $\sim_N$*-equivalences and bisimulation*

1. *For any LTS,* $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$, *the following property holds :* $\forall N \in \mathbb{N}, \sim \subset \sim_{n+1} \subset \sim_n$
2. *If moreover,* $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *is finitary then :*

$$\sim = \bigcap_{N\geq 0} \sim_N$$

3. *If moreover,* $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *is finite then :* $\sim = \sim_{n_s}$ *where* $n_s = |S|$

1. By recurrence on n
2. Let us call $R = \cap_{N\geq 0} \sim_N$.
   According to the first point of the proposal, $\sim \subset R$. To show that $R \subset \sim$, knowing that $\sim$ are the union of the bisimulations, it is enough to prove that R is a bisimulation. Let $s \; R \; s'$ and $s' \stackrel{a}{\rightarrow} t'$. Then by definition of $R, \forall n, \exists t_n$ such that $s \stackrel{a}{\rightarrow} t_n$
   and $t' \sim_n t_n$ Since the system of transitions is finitary, $\exists t$ such that $t = t_N$ for an infinity of n. What means that $\forall n, \exists n' > n$ such that $t' \sim_{n'} t$. What implies according to the first point that $t' \sim_N t$ thus $t' \; R \; t$. The second part of the proof is similar to the first.

3. According to the first point from the proposal, $\sim_{n+1} \subset \sim_N$. Moreover if for a system of transitions (finite or infinite) $\sim_{n+1} = \sim_N$, then $\sim_{n+1} = \sim$ since it comes while replacing in the definition $\sim_N$ by $\sim_{n+1}$, that $\sim_{n+1}$ is a relation of bisimulation. Finally let us suppose that for S, all the relations $\sim_I$ per $0 \leq I \leq n_s$ are different, then the number of classes of equivalence grows strictly according to i, which is absurd since this number must be lower or equal to n_s. There thus exists $n \leq n_s$ such that $\sim_n = \sim$ and consequently $\sim_{n_s} = \sim$.

The last characterization (cf prop 32.3) is particularly important since it provides us an algorithm to decide the bisimulation in the case of finite LTS.

Property 33 *Application, Equivalence and Quotient set*

Let f be an application of $A \mapsto B$:

1. *Let $\equiv_f$, the binary relation $\subset A \times A$, defined by $a_1 \equiv_f a_2 \sigma f(has) = f(b)$.*
   $\equiv_f$ is an equivalence relation
2. *Is $\pi^f(A) =_{Def} \bigcup_{B \in f(A)} f^{-1}(b)$.*
   $\pi^f(A)$ define a partition of A.
3. *$\pi^f(A) = A/\equiv_f$*

1) is obvious since âĂIJ=âĂİ is itself an equivalence relation. 2) Π_A is a covering of A of whose âĂIJblocksâĂİ are disjoined since f^{-1} is injective. 3) is assured because $\forall \pi \in \pi^f(A)$: a_1 and $a_2 \in \pi \Rightarrow a_1 \equiv_f a_2$

Definition 34 *Output of a state*

For $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, one consider the application $Output_{\mathcal{LTS}}$: $S \mapsto \mathcal{P}(\Sigma)$ which associates with each state $q \in S$, the subset of Σ defined in the following way : $Output_S(s) =_{Def} \{a \in \Sigma$ such that $s \stackrel{a}{\rightarrow}\}$

When there is no ambiguity on the LTS, one will note simply $Output(S)$ in the place of $Output_{\mathcal{LTS}}(S)$

Property 35 *Equivalent characterization of the $\sim_1$-equivalence*

By taking again the notations introduced into the property 33, one considers also the relation of equivalence $\equiv_{Output}$.

For any LTS, $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, we have : $\sim_1 = \equiv_{Output}$

Two states are equivalent with order 1 if they allow to carry out the same actions. It is enough to notice that $\sim_1$ is defined starting from $\sim_0$ for which two states are always equivalent (i.e $\sim_0 = S \times S$)

The property 35 thus makes it possible to directly compute the set of the classes of equivalence of S for $\sim_1$ (in other words, the quotient of S by $\sim_1$), by using the partition of S defined by the application $Output^{-1}$. One second obvious property which can be made profitable to limit computation consists in noticing that the relation of bisimulation (and more generally any equivalence of behavior) cannot distinguish two deadlock states.

Property 36 *Bisimulation and Deadlock*

For any LTS, $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$*, and any pair of states* $p, q \in S$

$[\ Output(p) = \ Output(q) = \emptyset] \Rightarrow p \sim_n q, \forall n \in \mathbb{N}$

Example 2 Application to the coffee machines $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ represented 7 :

We want to know if $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ are bisimilar.

As the sets of respective states of these machines (S and S') are disjoined, we pose $\mathcal{S} = S \cup S'$ and we will seek the largest bisimulation (i.e $\sim$) contained in $\mathcal{S}$. $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ will be bisimilar if $0 \sim 0'$. We calculate $\sim$ by considering the sequence of the $\sim_K$-equivalences (cf def 31 and prop 32).

Computation of $\sim_1$

The table below represents the graph of the application $Output^{-1}$ (def 34), by using the property 35, one obtains
$\mathcal{S}/\sim_1 = \{\{0, 0'\}, \{\{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$

$\mathcal{P}(\Sigma)$	$\mathcal{P}(\Sigma) \mapsto S$	$\mathcal{P}(\Sigma) \mapsto$	$\mathcal{P}(\Sigma) \mapsto S\cup$
Σ	$\emptyset$	$\emptyset$	$\emptyset$
{Tea, Coffee}	$\{1\}$	$\emptyset$	$\{1\}$
{Tea, Coin}	$\emptyset$	$\emptyset$	$\emptyset$
{Coffee, Coin}	$\emptyset$	$\emptyset$	$\emptyset$
{Coin}	$\{0\}$	$\{0'\}$	$\{0, 0'\}$
{Coffee}	$\emptyset$	$\{1'\}$	$\{1'\}$
{Tea}	$\emptyset$	$\{3'\}$	$\{3'\}$
$\emptyset$	$\{2, 3\}$	$\{2', 4'\}$	$\{2, 2', 3, 4'\}$

Calcul of $\sim$

$0 \not\sim_2 0'$: Indeed $0 \stackrel{\text{Coin}}{\rightarrow} 1$ and none the successors of 0 ' by Coin (i.e 1 ' and 3 ') is equivalent to order 1 to 1 (i.e $1' \not\sim_1 1$ and $3' \not\sim_1 1$). One can thus directly deduce that $\langle D, 0\rangle$ and $\langle Of, 0'\rangle$ are not bisimilar.

The property 36, ensures that $\{2, 2', 3, 4'\} \in \mathcal{S}/\sim$ Except the class $\{2, 2', 3, 4'\}$, the other classes are reduced to a singleton and thus minimal. One can thus deduce from it that $\mathcal{S}/\sim = \mathcal{S}/\sim_2 = \{\{0\}, \{0'\}, \{\{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$

Operational characterization

The process describes in the definition 4.1.1 makes it possible to obtain the largest bisimulation included in a given binary relation R like the limit of the decreasing sequence of relations $<\sim_N>_{N\geq 0}$. Each term of the sequence can be described in an equivalent way in the form of a partition of the set of the states. The computation of the terms of this decreasing sequence returns to the problem of refinement of a partition (Multi Relational Coarset Problem Partition) [PT 87] used initially within the framework of automata minimization [AHO 74]. One thus obtains the most powerful algorithms in $O(\Delta.log(S))$ where Δ denotes the number of transitions and S the number of states of the graph [FER 89].

Π-Bisimulation

We saw in this section, the standard relation of bisimulation such as that presented by [PARK 81, MIL 89]. This one takes into account only the labels of events and does not allow to take into account the states of the system. The concept of Π-Bisimulation [CLE 89] generalizes the concept of bisimulation while imposing that the relation of bisimulation is contained in a relation of equivalence given a priori. One thus can, by the means of this relation, take into account certain characteristics of the states in the computation of the bisimulation. We will use this concept of bisimulation in section 4.3.1 to be able to consider an extension of Hennessy-Milner logic taking into account atomic propositions and also in section 4.2.5 to make sensitive observational equivalence to the divergence.

Definition 37 Π-*Bisimulation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *be a LTS and* Π *a relation of equivalence on* S *(i.e* $\pi \subset S \times S$*), a binary relation* B *on* S *is a relation of* Π*-bisimulation if it verifies :* $B \subset \Pi$ *and*

$\forall (p,p') \in B and \forall T \in \Sigma$:

$$[\forall q \in S \ If \quad p \stackrel{t}{\rightarrow} q \ then \ \exists q' \in S \ : \quad p' \stackrel{t}{\rightarrow} q' \ and \ (q,q') \in B]$$
$$and \quad [\forall q' \in S \ If \quad p' \stackrel{t}{\rightarrow} q' \ then \ \exists q \in S \ : \quad p \stackrel{t}{\rightarrow} q \quad and \ (q',q) \in B]$$

Let us note that if $\pi = S \times S$, one find the standard concept of bisimulation. The procedure of general decision given in section 4.1.1 adapts to the Π-bisimulations. It is enough to initialize the sequence of equivalence relations while taking $\sim_0 = \Pi$.

4.1.2. *Simulation and Co-simulation*

As much the concept of bisimulation induces an equivalence relation on the LTS, as much the concept of simulation makes it possible to define an pre-order (a reflexive and transitive binary relation) on the LTS. the concept of simulation is defined by breaking the symmetry of the bisimulation definition.

Definition 38 *Simulation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$*, a binary relation* R*,* $(R \subset S\times)$ *is a simulation between* $\mathcal{LTS}$ *and* $\mathcal{LTS}'$ *if it verifies :* $\forall (p,p') \in R and \forall T \in \Sigma$:

$\forall q \in S$ *If* $p \stackrel{t}{\rightarrow} q$ *alors* $\exists q' \in S'$: $p' \stackrel{t}{\rightarrow} q'$ *et* $(q,q') \in R$

Extension to the ILTS : *As for the relation of bisimulation, the relation of simulation can be extended to the initialized labeled transition systems as follows :* $\langle \mathcal{LTS}, s_0\rangle$ *simulates* $\langle \mathcal{LTS}', s'_0\rangle$ *if there is a relation of simulation connecting their respective initial states : i.e., there exists a relation of simulation* R *containing* (s_0, s'_0)

The **Co-simulation** makes it possible to obtain a relation of equivalence from the simulation pre-order.

Definition 39 *Co-simulation*

$\mathcal{LTS}$ *Co-simulates* $\mathcal{LTS}' \Leftrightarrow_{Def}$ $\mathcal{LTS}$ *simulates* $\mathcal{LTS}'$ *and* $\mathcal{LTS}'$ *simulate* $\mathcal{LTS}$*.*

Example 3

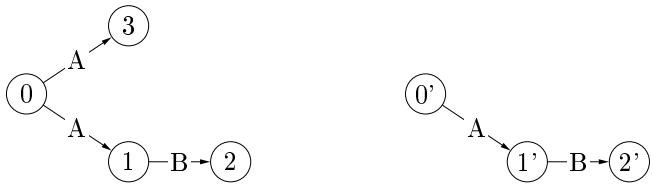


Figure 9: Two LTS co-similar and not bisimilar

Let us consider $R = \{(0,0'),(1,1'),(2,2'),(3,1')\}$ and check that it is indeed a simulation between $\langle S, 0\rangle$ and $\langle S', 0'\rangle$.

$0 \stackrel{A}{\rightarrow} \{1,3\}$ and $0' \stackrel{A}{\rightarrow} 1'$ with $(1,1') \in R_1$ and $(3,1') \in R_1$
$1 \stackrel{B}{\rightarrow} 2$ and $1' \stackrel{B}{\rightarrow} 2'$ with $(2,2') \in R_1$

Like 2 and 3 are deadlocks, it has anything more to check.
R_1 is thus a simulation containing $(0,0')$, therefore $\langle S,0\rangle$ simulates $\langle S',0'\rangle$.

In the other direction, one shows just as the relation $R_2 = \{(0',0),(1',1),(2',2)\}$ is a relation of simulation between $\langle S',0'\rangle$ and $\langle S,0\rangle$. Indeed, we have : $0' \stackrel{A}{\rightarrow} 1'$ and $0 \stackrel{A}{\rightarrow} 1$ with $(1',1) \in R_2$ & $1' \stackrel{B}{\rightarrow} 2'$ and $1 \stackrel{B}{\rightarrow} 2$ with $(2',2) \in R_2$

Theen $\langle S,0\rangle$ simulates $\langle S',0'\rangle$ and finally and $\langle S,0\rangle$ and $\langle S',0'\rangle$ are thus Co-similar.

On the other hand, $\langle S,0\rangle$ and $\langle S',0'\rangle$ are not bisimilar.

It is enough to reason by the absurb and to consider B a relation of bisimulation. This one has minimum would contain $(0',0)$. Consequently, like $0 \stackrel{A}{\rightarrow} \{1,3\}$ and $0' \stackrel{A}{\rightarrow} 1'$, B should also also contain the pairs $(1',1),(3',1)$. As $1' \stackrel{B}{\rightarrow} 2$ and $3 \stackrel{B}{\not\rightarrow}$ one cannot have $(1',3) \in B$ from where contradiction appears.
nb : The relation $B' = \{(3,2'),(2,2')\}$ is the largest bisimulation between S and S'.

Remark the preceding example shows that Co-simulation is weaker than the bisimulation. The bisimulation can however be defined in terms of simulation in the following way : a relation of simulation R of which the symmetrical relation R^{-1} is itself a relation of simulation is one bisimulation. The section **??** modalcarac presenting characterization modal equivalences of behavior will enable us to specify the relations between concepts of simulation, Co-simulation and bisimulation.

4.1.3. *Procedure of decision for simulation*

Construction presented is very close to that presented in 4.1.1 to decide bisimulation. Instead of using a succession of relations (the $\sim_K$-equivalences of the definition 31), one introduces a function E which by successive iterations will make it possible to obtain its smaller fixed point which corresponds, in fact, with the relation of sought simulation. The interested reader will find a talk complete of this construction and associated proofs in [ARN 92].

We present the more general form here allowing to compute one Π-Simulation similar to the concept of Π-bisimulation introduced in definition 37). This computation built starting from an arbitary binary relation R, the greatest simulation (if it exists) contained in R.

Definition 40 *Simulation generated by a relation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *be two LTS*

We consider the mapping $E : \mathcal{P}(S\times) \mapsto \mathcal{P}(S\times)$ *which associates with any binary relation* $R \subset S \times S'$*, the relation* $E(R)$ *on* $S \times S'$ *defined as follows :*
$(s, s') \in E(R) \Leftrightarrow_{Def} (1) \wedge (2)$ *where*
(1) $(s, s') \in R$
(2) $\forall t \in \Sigma, \forall q \in S \ : \ s \stackrel{t}{\rightarrow} q \Rightarrow \exists q' \in S' \ : \ s' \stackrel{t}{\rightarrow} q'$ *such that* $(q, q') \in R$

Property 41 *Characterization of a simulation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *be two LTS*

1. *the following property holds :*
   $\forall N \in \mathbb{N}, E^{n+1}(R) \ \subset E^N(R) \ \subset R$
2. *If moreover,* S *and* S' *are finitaries then :*
   The sequence of relations $_{N\geq 0}$ *admits for limit* R^ω *with*
   $R^\omega = \bigcap_{N\geq 0} E^N(R)$.
   R^ω *is the greatest fixed point of the function* E
   i.e $E(R^\omega) = R^\Omega$ *and* $E(A) = A \Rightarrow A \subseteq R^\omega$
3. *If moreover,* S *and* S' *are finite then :*
   $R^\omega = E^K(R)$ *where* $k = max(|S|, |S'|)$

The preceding property is similar to the characterization of the bisimulations (cf prop 32). Item 2) shows that R^ω is the greatest simulation between Σ and Σ' included in R. As E is a decreasing application of a powerset in itself, the convergence of the sequence is assured [ARN 92]. By construction, R^ω is included in R and constitutes the greatest solution of the equation $E(R) = E$, R^ω is thus the largest simulation included in R. Item 3) provides a means to decide simulation between two finite LTS.

4.2. *Weak Equivalences*

The relations of equivalence which we considered until now supposed that the transitions systems that we compare admitted the same sets of transitions labels.

At this point, this constraint of identity of the alphabets of action strongly limits the possibilities of us of equivalences or the pre-orders of behaviour : it is not possible, for example to compare systems described at various levels of abstraction. Thus the various drink distributers encountered in this section only represent in an abstract way the "service" rendered by a drink distributer ;

obviously a true distributer would be more complex. In practice, the behavioral approach thus requires to be able to compare systems described at various levels of abstraction. Concretely, one wants to compare systems by making "abstraction" of certain events (actions) which are not relevant with respect to the analysis that one wants to lead. The first step consists in defining the criterion of observation of the system : the observed events and those whose one makes abstraction.

A simple solution consists in considering a subset $\mathcal{O}$, observable actions, of the set of the labels Σ. Once this defined criterion, it is necessary to clarify what one understands by disregarding inobservable event. A first alternative is provided to us by re-using the concept of projection already defined in the language theory.

Definition 42 *Projection of a language*

Let $\mathcal{O}$ be a subset of Σ, σ a word of Σ^*, the projection of σ out of $\mathcal{O}$, noted $\sigma_{\lfloor\mathcal{O}}$ is recursively defined by :

$$\lambda_{\lfloor\mathcal{O}} =_{Def} \lambda \text{ and } (\sigma.a)_{\lfloor\mathcal{O}} =_{Def} \begin{cases} \sigma_{\lfloor\mathcal{O}}.a & \text{if } a \in \mathcal{O} \\ \sigma_{\lfloor\mathcal{O}} & \text{sinon} \end{cases}$$

Projection operates as a "gum" which erases all the letters of the word not belonging to $\mathcal{O}$. This operator of projection extends in a canonical way to a set of words : $L \subset \Sigma^*$: $L_{\lfloor\mathcal{O}} =_{Def} \{\sigma_{\lfloor\mathcal{O}} \ : \sigma \in L\}$

Definition 43 *Weak language equivalence*

The comparison between the systems is defined with respect to a common criterion of observation commun : these systems will be equivalent if projections of their languages compared to this observation criterion are equal.

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ and $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ two transitions systems, s_0 and s'_0 their respective initial states and $\mathcal{O}$ a common criterion of observation i.e., $\mathcal{O} \subset \Sigma \cap \Sigma'$

$$\langle \mathcal{LTS}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{LTS}', s'_0\rangle \sigma L(\langle \mathcal{LTS}, s_0\rangle)_{\lfloor\mathcal{O}} = L(\langle \mathcal{LTS}', s'_0\rangle)_{\lfloor\mathcal{O}}$$

nb : This equivalence generalizes equivalence language presented definition 18. Indeed, by taking $\mathcal{O} = \Sigma \cup \Sigma'$, then $\langle \mathcal{LTS}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{LTS}', s'_0\rangle \sigma \langle \mathcal{LTS}, s_0\rangle \equiv \langle \mathcal{LTS}', s'_0\rangle$

Example 4 Application of weak equivalence language

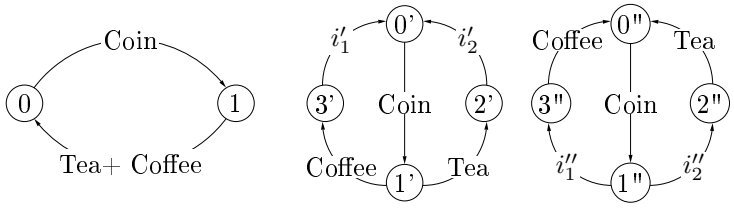


Figure 10: Three other coffee machines

Let us consider the three coffee machines $\langle M, 0\rangle, \langle M', 0'\rangle$ and $\langle M'', 0''\rangle$ represented Figure [2] 10 and compare by observing only the alphabet $\mathcal{O}$ made up of the actions Coin, Tea and Coffee.

These three systems admit after projection the same language describes by the following rational expression : (Coin.(Tea + Coffee))*, they are thus equivalent language for $\mathcal{O}$: $\langle M, 0\rangle \equiv_{\mathcal{O}} \langle M', 0'\rangle \equiv_{\mathcal{O}} \langle M'', 0''\rangle$.

For as much from the point of view of a customer, $\langle M'', 0''\rangle$ is distinguished from both others since this machine "chooses in an autonomous way" the drink delivered with the customer. In state 1", M " is ready to offer Tea or of Coffee but the actions i''_1 and i''_2 whose one had a priori wanted to make abstraction have an influence on the service offered by these machines since according to their occurrences, Tea or Coffeewill be delivered.

The actions whose one had decided a priori to make abstraction ($\{i'_1, i'_2, i''_1, i''_2\}$) disappear under the effect of the "gum" which operated during projection. For some of between they, i'_1 and i'_2, the abstraction carried out is legitimate in the sense that those do not modify the "observable" behavior of the system, for others i''_1 and i''_2, the abstraction carried out is not founded since these actions have an observable influence on the behavior of the system.

All the difficulty is to know a priori if it is "reasonable" to hide an action. The observational equivalence, introduced by R. Milner into his calculation for communicating systems CCS [MIL 89], introduces a concept of "abstract experiment" which offers a solution to this problem.

4.2.1. *Experiment, Saturation*

The concept of abstract experiment allows an abstraction less radical than the concept of projection which erases purely and simply any inobservable action. Here, the inobservable actions are initially renammed by a common

[2] label "Tea+ Coffee" connecting states 1 to 0 means simply that one can go from state 0 to state 1 either by the Tea action or by the action Coffee

symbol τ. Moreover, one new relation of transition, known as "abstract expreimentation" and noted $\Rightarrow$, is defined by taking of account the relation of original transition ($\rightarrow$) and the inobservable sequences of actions. We give the definition now and will give the intuition of this one of it by studying the example of saturation presented Figure 5.

Definition 44 *Abstract experiment :* $\Rightarrow$

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *be a LTS,* $\mathcal{O}$ *a subset of* Σ *and* τ *a symbol not belonging to* Σ

$\Rightarrow \subset S \times \Sigma \cup \{\tau\} \times S$ *can be defined as follows :* $\Rightarrow =_{Def} \stackrel{\tau}{\Rightarrow} \bigcup \cup_{o\in\mathcal{O}} [\stackrel{o}{\Rightarrow}]$ *where*

- $\stackrel{\tau}{\Rightarrow}$ *the relation of transition relating to the inobservable experiments, is obtained by taking the reflexive and transitive closure of the union of transition inobservable relations :*

  $\stackrel{\tau}{\Rightarrow} =_{Def} [\bigcup_{i\in\Sigma\setminus\mathcal{O}} \xrightarrow{i}]^*$

  nb : $\stackrel{\tau}{\Rightarrow}$ *renames all the inobservable labels with a common label* τ*.*

  The transitivity of $\stackrel{\tau}{\Rightarrow}$ *makes it possible to consider a sequence of inobservable actions as an "atomic" action inobservable. Finally the reflexivity of* $\stackrel{\tau}{\Rightarrow}$ *ensures that of any state of the LTS it is possible to make an action inobservable : it only consists to add an inobservable "neutral" transition buckling on any state of the LTS.*

- $\stackrel{a}{\Rightarrow}$*, the relation relating to the observable experiments is defined as the double composition (on the right and on the left) of the inobservable relation of experiment* $\stackrel{\tau}{\Rightarrow}$ *with the original relation observable transition* $\xrightarrow{a}$*.*

  $\stackrel{a}{\Rightarrow} =_{Def} \stackrel{\tau}{\Rightarrow} o \xrightarrow{a} o \stackrel{\tau}{\Rightarrow}$ *for* $a \in \mathcal{O}$

  nb : $\stackrel{a}{\Rightarrow}$ *makes it possible to extend concept of observable transition by integrating the sequences of inobservable transitions. As* $\stackrel{\tau}{\Rightarrow}$ *is reflexive,* $\stackrel{a}{\Rightarrow}$ *includes the original relation observable transition :* $\xrightarrow{a}$*. The double composition on the right and on the left makes it possible to regard as one observable action "atomic" any preceded observable action and followed by a sequence of inobservable actions.*

Definition 45 *Saturation of a labelled transition system*

The LTS obtained by substituting the experiment relation ($\Rightarrow$*) to the original transition relation (*$\rightarrow$*) is called LTS saturated.*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *be a LTS and* $\mathcal{O}$ *a subset of* Σ *and* τ *a symbol not belonging to* Σ

$Sat_{\lfloor \mathcal{O}}(\mathcal{LTS}) =_{Def} \langle \Sigma \cup \{\tau\}, S, \{\stackrel{a}{\Rightarrow}\}_{a \in \Sigma \cup \{\tau\}} \rangle$

nb : If $\Sigma = \mathcal{O}$, saturation simply consists in adding to the initial system a âĂIJloopâĂİ labelled by τ on each state.

$Sat_{\lfloor \Sigma}(\mathcal{LTS}) = \langle \Sigma \cup \{\tau\}, S, [\{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \cup \{p \stackrel{\tau}{\rightarrow} p \ : p \in S\} \rangle$

Example 5 Example of Saturation

The figure 11 presents the output of the saturation applied to the systems M' and M " represented Figure 10 when $\mathcal{O} = \{\text{Coin, Tea, Coffee}\}$.

Not to overload the figure, the loops of τ resulting from closure reflexive of the relation $\stackrel{\tau}{\Rightarrow}$, normally associated with each state, are not not represented.

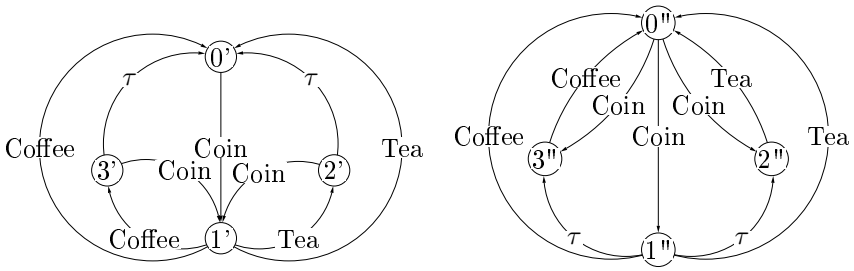


Figure 11: $Sat_{\lfloor \mathcal{O}}(M')$ et $Sat_{\lfloor \mathcal{O}}(M'')$

The inobservable labels of actions $(i'_1, i'_2, i''_1, i''_2)$ were renamed in the saturated systems. This renaming and the action of the double composition of inobservable actions, led to a non deterministic relation of observable experiment $\stackrel{O}{\Rightarrow}$.

The saturation may furnishe a non-deterministic LTS from a deterministic LTS. Thus from 1' in $Sat_{\lfloor \mathcal{O}}(M')$, the experiment Coffee will lead into 3' or 1' according to whether only the action Coffee occurred or that this one was followed by the inobservable action i'_1. In the same way from state 0" in $Sat_{\lfloor \mathcal{O}}(M'')$, the experiment Coin can lead in state 1" where the actions Tea and Coffee are possible, in state 2" where only the action Tea is possible or in state 3" where only the action Coffee is possible.

The abstraction carried out while choosing not to observe the actions i''_1, I''_2 does not occult the fact that the machine M'' can choose drink in the place of the customer.

4.2.2. *Weak bisimulation, Observational Equivalence*

The observational equivalence of two systems [MIL 89] can be defined directly starting from the bisimulation (def 25) by considering the relation of bisimulation between the saturated systems.

Definition 46 *Weak bisimulation*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *be two LTS and* $\mathcal{O} \subset \Sigma \cap \Sigma'$*, a set of common labels of actions.*

A binary relation $B \subset S \times S'$ *is a* $_{\lfloor\mathcal{O}}$*Bisimulation between* $\mathcal{LTS}$ *and* $\mathcal{LTS}'$ *if* B *is a bisimulation between* $Sat_{\lfloor\mathcal{O}}(\mathcal{LTS})$ *and* $Sat_{\lfloor\mathcal{O}}(\mathcal{LTS}')$

This bisimulation parameterized by a set of observable actions is often described as âĂIJweakâĂİ bisimulation in opposition to the standard bisimulation, called âĂIJstrongâĂİ, which takes into account all the labels of actions. The observational equivalence introduced into CCS [MIL 89] is the largest weak bisimulation.

The concepts of strong and weak bisimulation coincide when all the labels of actions are observed (i.e $\mathcal{O} = \Sigma \cup \Sigma'$) : B is a bisimulation (strong) between $\mathcal{LTS}$ and $\mathcal{LTS}'$ if B is a $_{\lfloor(\Sigma\cup\Sigma')}$B*isimulation* between $Sat_{\lfloor\Sigma}(\mathcal{LTS})$ and $Sat_{\lfloor\Sigma'}(\mathcal{LTS}')$

4.2.3. *Decision of the weak bisimulation*

As the weak bisimulation is finally defined like a strong bisimulation on saturated systems, the procedure of decision which we gave for the strong bisimulation applies to decide weak bisimulation.

When the saturated system is considered, the property 35 remains valid (it can be simplified by noticing that any state of the saturated system has τ in its *Output*). It is the same for the property 36 but this one becomes inoperative in the saturated system, any state has at least one τ-successor (itself). In the case of the weak bisimulation, the property 36 is reformulated in terms of weak blocking (def 47).

Property 47 *Weak bisimulation and weak blocking*

For any LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$*, any subset* $\mathcal{O}$ *of* Σ *and any pair of states* $p, q \in S$

$[\ Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{LTS})}(p) = \ Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{LTS})}(q) = \{\tau\}] \Rightarrow p \sim_{\mathcal{O}} q$

Example 6 Example (continued) 5

One considers again the ILTS $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$ of the example 5. One seeks a bisimulation between $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$. Let $\mathcal{S} = S \cup S'$

The property 36 enables us to obtain equivalence at order 1 :

$$\mathcal{S}/\sim_1 = \{\{0', 2', 3', 0''\}, \{1', 1''\}, \{2''\}, \{3''\}\}$$

At order 2, it is easy to see that $1' \not\sim_2 1''$. Indeed, there are $1'' \stackrel{\tau}{\Rightarrow} 2''$ whereas 1' admits only one τ-successor, 1' itself, which is not equivalent at order 1 to 2". While thus continuing, one would show, with order 3, that 1' and 1" are not equivalent. Thus $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$ are not observationnaly equivalent.

4.2.4. *Abstract/Quotient model*

Equivalences or the pre-orders which we saw up to now allow to compare a system and its specification, both expressed in the form of graph. The behavioral approach, proceeding by comparison, allows to be sure that the system and its specification have the same properties (the same behavior) modulo the criterion of abstraction selected to perform the comparison.

For equivalences relations, one can also proceed by projection or âĂIJintrospectionâĂİ. Instead of comparing two LTS, one can build the smallest equivalent LTS [3].One speaks then of projection or of abstract model.

Definition 48 *Observational projection*

For a LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$*, a subset* $\mathcal{O} \subset \Sigma$ *from observable labels and* $\sim_{\mathcal{O}}$*, the associated observational equivalence relation, one note* $\mathcal{LTS}/\sim_{\mathcal{O}}$ *the quotient of* $\mathcal{LTS}$ *by* $\sim_{\mathcal{O}}$

$\mathcal{LTS}/\sim_{\mathcal{O}} =_{Def} \langle S/\sim_{\mathcal{O}}, \mathcal{O} \cup \{\tau\}, \{\stackrel{a}{\rightsquigarrow}\}_{a \in \mathcal{O} \cup \{\tau\}} \rangle$ *where :*

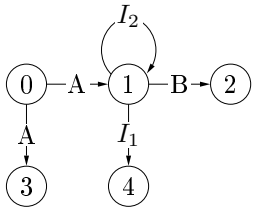
1. $S/\sim_{\mathcal{O}}$ *is the quotient of* S *by* $\sim_{\mathcal{O}}$
2. $\stackrel{a}{\rightsquigarrow}$ *is the smallest relation verifying :*
   (a) $(a \in \mathcal{O}$ *and* $q \stackrel{o}{\Rightarrow} q') \Rightarrow q/\sim_{\mathcal{O}} \stackrel{o}{\rightsquigarrow} q'/\sim_{\mathcal{O}}$
   (b) $(a \notin \mathcal{O}$ *and* $q \stackrel{o}{\Rightarrow} q'$ *and* $q \not\sim_{\mathcal{O}} q') \Rightarrow q/\sim_{\mathcal{O}} \stackrel{\tau}{\rightsquigarrow} q'/\sim_{\mathcal{O}}$

(2.a) means that any observable transition remains in projection. (2.b) means that only the inobservable transitions connecting two non equivalent states remain in projection.

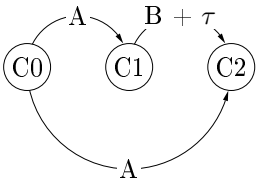
[3] with respect to the number of states

Example 7 Example of projection

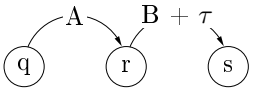
One considers $\langle X, 0\rangle$ the ILTS represented opposite. One chooses to observe $\mathcal{O} = \{A, B\}$. In this case, $X/\sim_{\mathcal{O}} = \{\{0\}, \{1\}, \{2, 3, 4\}\}$



the ILTS $\langle X/\sim_{\mathcal{O}}, C0\rangle$ obtained by projection is represented opposite. It comprises 3 states : $C0 = \{0\}, C1 = \{1\}, C2 = \{2, 3, 4\}$ One can note that certain inobservable transitions remain ($1 \xrightarrow{I_1} 4$ which materialized the possibility of blocking after A appear in the form $C1 \overset{\tau}{\rightsquigarrow} C2$) while others disappear. The transition $1 \xrightarrow{I_2} 1$ watch which the system can have an inobservable infinite execution, one will speak thereafter (cf section 4.2.5) about divergence. This possibility of divergence of the system is absorbed in the class $C1$).



By construction, the ILTS obtained by projection ($\langle X/\sim_{\mathcal{O}}, C0\rangle$) is equivalent to the ILTS projected $\langle X, 0\rangle$. Projection is minimal with respect to the number of states. On the other hand it is not minimal with respect to the number of transitions. Consider $\langle Y, Q\rangle$, the ILTS represented opposite. There are again $\langle Y, Q\rangle \sim_{\mathcal{O}} \langle X, 0\rangle$ but $\langle Y, Q\rangle$ comprises less transitions than $\langle X/\sim_{\mathcal{O}}, C0\rangle$.



Example 8 Continuation of the example 5

By taking again the results obtained in the example 5, we have :

$$\mathcal{S}/\sim = \{\{0', 2', 3'\}, \{0''\}, \{1'\}, \{2''\}, \{3''\}\}$$

One can by simple projection [4] to deduce the relation $\sim$ from each ILTS :

$$S/\sim = \{\{0', 2', 3'\}, \{1'\}\} \text{ and } S'/\sim = \{\{0''\}, \{1''\}, \{2''\}, \{3''\}\}$$

The figure below shows the various steps of the computation : on the left the initial system, in the medium the saturated system and, on the right, the projected system.

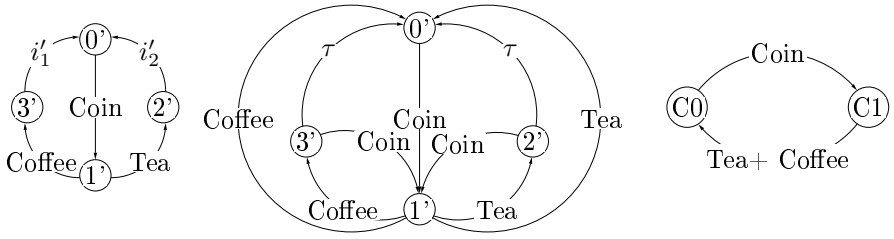


Figure 12: $\langle M', 0' \rangle, Sat_{\lfloor \mathcal{O}}(M')$ and $\langle M'/\sim_{\mathcal{O}}, 0'/\sim_{\mathcal{O}} \rangle$

For $\langle M'', 0'' \rangle$, classes of equivalence, $S''/\sim_{\mathcal{O}}$, are reduced to singletons, and the LTS projected is identical (isomorph except for the renaming of the inobservable actions i''_1 and I''_2 by τ) with the initial LTS.

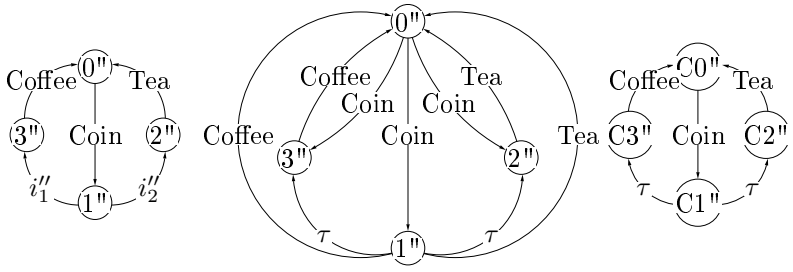


Figure 13: $\langle M'', 0'' \rangle, Sat_{\lfloor \mathcal{O}}(M'')$ and $\langle M''/\sim_{\mathcal{O}}, 0''/\sim_{\mathcal{O}} \rangle$

The projection of $\langle M'/\mathcal{O}, 0'/\mathcal{O} \rangle$ give again the machine $\langle M, 0 \rangle$ (cf 6) ; these two LTS are indeed observationaly equivalent.

For $\langle M', 0' \rangle$, the fact of obtaining only one class gathering states 0 ', 2 ' and 3 ' allows to âĂIJinteriorizeâĂİ the two transitions (i'_1 and i'_2) whose one had decided to make abstraction. The computation of observational equivalence

[4] projection here in the usual sense : projection of the partition on the sets of states respectively associated with each one of the LTS

allows to legitimate a posteriori this choice : occurrence of these events do not modify the behavior 'observed" of the system.

For $\langle M'', 0'' \rangle$, ignore the actions i_1'' and I_2'' does not have sense since according to their occurrences the customer will have or will not have the choice of its drink.

4.2.5. *Observational equivalence and divergence*

The example precedent has shown that observational equivalence performs an abstraction relatively reasoned with respect to the hidden events. Contrarily with the weak language equivalence which a priori gums all hidden events, observational equivalence can preserve "visible" events that one did not want to a priori observe ; typically the case of the events inobservable which remains present on the LTS quotient (unobservable events connecting not equivalent states)

Among the important concepts which "are masked" by observational equivalence, we find the concept of divergence, already met in our example of projection 7. In the case of finite LTS, the divergence is directly related to the existence of cyclic paths labelled by inobservable labels, which we will call of the τ-cycles.

The fact of observing only certain events leads us to refine the standard concept of state of blocking. Indeed, we want distinguish the states for which the system (without being blocked) may perform only inobservable actions (weak blocking states) and of the states where the system can carry out an infinite number of inobservable actions (divergent states).

Definition 49 *Weak blocking, divergence*

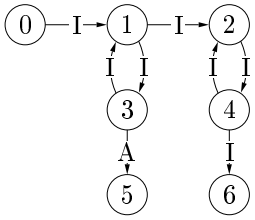
For a LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, $\mathcal{O}$ *a subset of* Σ *and* s *a state of* S

1. s *is a weak blocking for* $\mathcal{O}$ *if* $Output_{Sat_{\lfloor \mathcal{O}}(\mathcal{LTS})}(s) = \{\tau\}$
2. s *is a state of divergence for* $\mathcal{O}$ *if* $\exists \sigma \in L(\langle S, s \rangle) \cap (\Sigma \setminus \mathcal{O})^{\infty}$

A weak blocking corresponds in a state where the system cannot evolve in an observable manner (the only actions carried out are not observable) an observer cannot thus make the difference between a state of weak blocking and a state of blocking. Contrary to a weak blocking, from a state of divergence the system can evolve in an observable way but it can also evolve indefinitely in an inobservable way.

Example 9

One considers $\langle L, 0\rangle$ represented opposite. Only the action A is observed ($\mathcal{O} = \{A\}$).
5 and 6 is states of blocking,
2, 4, 5, and 6 are states of weak blocking,
0, 1, 2, 3 and 4 are states of divergence.



Definition 50 *τ-cycles*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ be a LTS and $\mathcal{O} \subset \Sigma$ a subset of observable labels.

Two states $s_1, s_2 \in S$ are connected by one τ-cycle if
$\exists \sigma_1 \in (\Sigma\backslash\mathcal{O})^*, \exists \sigma_2 \in (\Sigma\backslash\mathcal{O})^*$ such that $s_1 \stackrel{\sigma_1}{\rightarrow} s_2$ and $s_2 \stackrel{\sigma_2}{\rightarrow} s_1$

Property 51 *τ-cycle and weak bisimulation*

Two states $s_1, s_2 \in S$ belonging to same a τ-cycle are obviously bisimilar.

It is enough to notice that states s_1 and s_2 admit exactly the same derived states i.e : $\forall t \in \mathcal{O}\cup\{\tau\}, \forall s \in S$: $s_1 \stackrel{t}{\Rightarrow} s \Leftrightarrow s_2 \stackrel{t}{\Rightarrow} s$.

In terms quotient of LTS, a corollary of this property is that if a pair of states s_1 and s_2 is connected by a τ-cycle then all the elementary transitions constituting this τ-cycle are "hidden" inside the class of equivalence of s_1 (i.e that of s_2). In other words, an observational projection disregards all τ-cycle.

Example 10 Observational equivalence and divergence

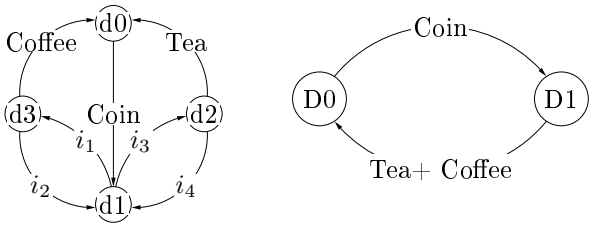


Figure 14: $\langle Div, 0\rangle$ and its quotient

Let us consider the coffee machine $\langle Div, C\rangle$ represented Figure 14. As until now $\mathcal{O} = \{\text{Coin}, \text{Tea}, \text{Coffee}\}$. The states $d1, d2$ and $d3$ are connected by one

τ-cycle and by applying property 51, it comes that $d1 \sim_{\mathcal{O}} d2 \sim_{\mathcal{O}} d3$. Consequently, $\sim_{\mathcal{O}} = \{\{d0\}, \{d1, d2, d3\}\}$. By noting $D_0 = \{d0\}$ et $D_1 = \{d1, d2, d3\}$, we obtain the LTS quotient. All the inobservable transitions belong to τ-cycles and dissapear into equivalence classes.

It was already shown (cf section 13) that the LTS $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ represented below were weak bisimilar. One would show in the same way that $\langle D, 0\rangle \sim_{\mathcal{O}} \langle Div, 0\rangle$. By transitivity, these 3 LTS are in weak bisimulation. For as much, their behaviors are strongly differents :

$\langle D, 0\rangle$ and $\langle D', 0'\rangle$ deliver "inevitably" one drink after it was paid, while for $\langle Div, 0\rangle$, the delivery of a drink is only âĂIJpotentialâĂİ.

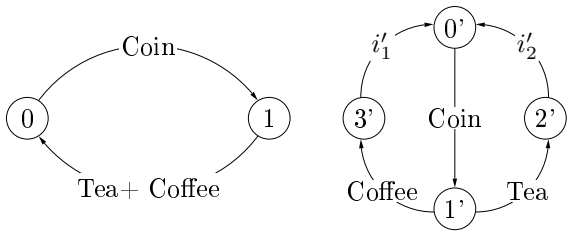


Figure 15: $\langle D, 0\rangle$ and $\langle D', 0'\rangle$: 2 LTS bisimilar with $\langle Div, do\rangle$

4.3. *Modal characterizations of behavioural equivalences*

The logical verification is based on properties expressed in a specific language (temporal logic for instance). Checking the system satisfies these properties is equivalent to show that the system is a model of these properties. The behavioral approach handles only behaviors. One studies equivalence between the behaviours of the system and of the specification and deduces, from this equivalence, that the system satisfies its specifications : however we have never explicitly stated the properties associated with the specification and never have specified the nature of the properties preserved by the used equivalence.

The work of M. Hennessy and R.Milner [HM 85] permits to understand the links between these two approaches of verification. It provides in particular a "logical" definition of the behavioral equivalences which specifies the type of properties that they allow to check.

4.3.1. *Definition of $\mathcal{HML}$*

In an intuitive way, a logic can be associated with a given behavioural equivalence (a logic in adequacy with an equivalence) so that the equivalent behaviors are the behaviors satisfying the same properties expressed in the adequate logic.

Definition 52 *$\mathcal{HML}$: Logic of Hennessy-Milner*

Syntax : *$\mathcal{HML}$ is the smallest set verifying :*
$true \in \mathcal{HML}$, $f, g \in \mathcal{HML} \Rightarrow f \wedge g, \neg g \in \mathcal{HML}$
$f \in \mathcal{HML}, a \in \Sigma \Rightarrow <a> f \in \mathcal{HML}$

Semantics : *The semantics of the formulas $\mathcal{HML}$ is defined with respect to a LTS $\mathcal{LTS}$. As for modal logics, one will note for $s \in S$ and $f \in \mathcal{HML}$: $\mathcal{LTS}, s \models f$ to indicate that the formula f is satisfied in the state s of the structure (here of the LTS) $\mathcal{LTS}$. As usually, the semantics of a formula of $\mathcal{HML}$ is defined by induction on the structure of the terms. In what follows : f and $g \in \mathcal{HML}$ and $a \in \Sigma$.*

$\models$, the relation of satisfaction, is the smallest relation verifying :
$\mathcal{LTS}, s \models true \quad \forall s \in S$
$\mathcal{LTS}, s \models f \wedge g$ *iff* $\mathcal{LTS}, s \models f$ *and* $\mathcal{LTS}, s \models g$
$\mathcal{LTS}, s \models \neg f$ *iff Not* $(\mathcal{LTS}, s \models \neg f)$
$\mathcal{LTS}, s \models <a> f$ *iff* $\exists s' \in S$ *such that* $s \xrightarrow{a} s'$ *et* $\mathcal{LTS}, s' \models f$

Abbreviations : $false \equiv \neg true$, $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $[a] f \equiv \neg <a> \neg f$
$<\sigma> f \equiv <a_1><a_2> \ldots <a_n> f$ *pour* $\sigma = a_1.a_2. \ldots a_n$

The semantics of a formula of $\mathcal{HML}$ (i.e $\models$, the relation of satisfaisability) is defined as for modal logics by regarding the LTS as structure of Kripke. Two differences may be noticed : $\mathcal{HML}$ do not utilize of atomic propositions. More exactly, the set of propositional variables only contains the variable *true* which is true in any state. By taking again the notations introduced into the section 2 : $\mathcal{P} = \{true\}$ and $\nu(s) = \{true\}$: $\forall s \in S$. On the other hand, the relation of accessibility between the "worlds" of the structure is now labelled.

Example 11 Examples of properties expressed into $\mathcal{HML}$

$\mathcal{LTS}, S \models <a> true$ *An a-experiment is possible starting from s.*

$\mathcal{LTS}, s \models <a> (<b> true \wedge <c> true)$
From s, an a-experiment is possible driving in a state where a b-experiment and a c-experiment are both possible.

$\mathcal{LTS}, s \models [a] false$ *From s, no a-experiment is possible.*

One again considers the LTS $\langle D, 0\rangle, \langle D', 0'\rangle$ and $\langle D'', 0''\rangle$ represented figure 6 and the properties $F_I \ : I \in [1, 4]$ below. The table below gives the results of the evaluation of the formulas F_I for each one of the LTS.

$F_1 \equiv$<a> [b]F
$F_2 \equiv$<a> (<b> $T \wedge$ <c> T)
$F_3 \equiv$ [a](<b> $T \wedge$ <c> T)
$F_4 \equiv$<a> ((<b> $T \wedge$ [c]F) $\vee$ (<c> $T \wedge$ [b]F))

$\models$	F_1	F_2	F_3	F_4
$D, 0$	F	V	V	F
$D', 0'$	V	F	F	V
$D'', 0''$	V	V	F	F

4.3.2. *Modal Characterization of the bisimulation*

Theory of a state : For a logic $\mathcal{L}$, one notes $TH \ : S \mapsto \mathcal{L}$, the mapping which associates with a state s S the set of the properties f of $\mathcal{L}$ that it satisfies (its theory). $TH_{\mathcal{L}}(S) =_{Def} \{F \in \mathcal{L} \ : S \models F\}$

Property 53 *Hennessy-Milner's theorem [HM 85]*

$\mathcal{HML}$ characterizes modally the bisimulation. Two states are bisimilar if they satisfy the same properties of logic $\mathcal{HML}$.
$s \sim q iff TH_{\mathcal{HML}}(s) = TH_{\mathcal{HML}}(q)$

Example 12 Return on the examples 6, 7 and 5

The LTS of the figure 6 are not bisimilar. By taking again the formulas of the table 11, one can conclude that these CO are 2 to 2 not equivalent :
- F_3 only holds for D thus D is equivalent neither to D' nor with D''.
- F_4 only holds for D' thus D' is equivalent neither to D nor with D''.

The LTS of the figure 7 are not bisimilar. Let us consider the following formula of $\mathcal{HML}$: - $G \equiv$<Coin> ((<Coffee> <Sugar> T) $\wedge$ (<Coffee> [Sugar]F)). $M, 0 \not\models G$ while $M', 0' \models G$

The LTS of the figure 5 are not bisimilar. Let $f \equiv$ [Coin](<Coffee> $true \wedge <$ Tea> $true$). f can be formulated as follows : after any occurence of the action Coin there is always the possibility of carrying out the actions Coffee and Tea. $M', 0' \models f$ while $M'', 0'' \not\models f$.

4.3.3. *$\mathcal{HML}$ and atomic propositions*

As we saw logic $\mathcal{HML}$, in its original statement, is based only on the events, the relation of π-bisimulation (cf definition 37) can be used to allow to explicitly take into account the concept of states.

Instead of considering only one labelled transitions system, we consider a labelled Kripke's structure
$\mathcal{LKS} = \langle AP, \Sigma, S, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}, \nu\rangle$ where in each state of S, the valuation ν associates a set of propositional variables $\in 2^{AP}$ (cf definition 1).

Definition 54 $\mathcal{HML}(AP)$

One notes $\mathcal{HML}(AP)$ the extension of $\mathcal{HML}$ to a set of atomic propositions AP as follows :

Syntax : *$\mathcal{HML}(AP)$ is the smallest set verifying :*
$AP \subset \mathcal{HML}(AP)$, $f, g \in \mathcal{HML}(AP) \Rightarrow f \wedge g \in \mathcal{HML}(AP), \neg f \in \mathcal{HML}(AP)$
$f \in \mathcal{HML}(AP), a \in \Sigma \Rightarrow <a> f \in \mathcal{HML}(AP)$

Semantics : *The semantics of the formulas $\mathcal{HML}(AP)$ is defined with respect to a labelled Kripke's structure $\mathcal{LKS} = \langle AP, \Sigma, S, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}, \nu\rangle$*

$\models$, the relation of satisfaction, is the smallest relation verifying :
$\mathcal{LKS}, s \models P$ *iff* $P \in \nu(P)$
$\mathcal{LKS}, s \models f \wedge g$ *ssi* $\mathcal{LKS}, s \models f$ *et* $\mathcal{LKS}, s \models g$
$\mathcal{LKS}, s \models \neg f$ *ssi Non* $(\mathcal{LKS}, s \models \neg f)$
$\mathcal{LKS}, s \models <a> f$ *ssi* $s \stackrel{a}{\rightarrow} s'$ *et* $\mathcal{LKS}, s' \models f$

Property 55 *Modal characterization of the π-bisimulation*

By taking again the notations introduced into the definition 33, one considers the partition $\pi^\nu(S)$ of S defined by the application ν and one considers the associated π-bismulation (cf def 37).

$\mathcal{HML}(AP)$ gives a modal chatacterization for the $\pi^\nu(S)$-bisimulation : two states are $\pi^\nu(S)$- bisimilar if they satisfy the same formulas of logic $\mathcal{HML}(AP)$.

$$\forall p, q \in S : p \sim_{\pi^\nu} q \text{ ssi } TH_{\mathcal{HML}(AP)}(q) = TH_{\mathcal{HML}(AP)}(q)$$

Example 13 $\mathcal{HML}$ and divergence

We saw that observational equivalence masked the divergent evolutions (cf 4.2.5) : it follows that the concept of event inevitable is not exprimable in $\mathcal{HML}$.

By taking again notations of section 3.3, one notes $\langle \mathcal{M}, S\rangle \models \mathbf{AFX}_{\{T\}} true$ to mean that execution of event t is inevitable from the state s. The adequacy property of adequacy (prop 53) provides us a simple means for to show that $\mathbf{AFX}_{\{T\}} true$ can not be expressed in $\mathcal{HML}$.

Let us consider the LTS represented figure 16. $1 \models \mathbf{AFX}_{\{A\}} true$, $0 \not\models \mathbf{AFX}_{\{A\}} true$, obviously we have $1 \sim 0$ and consequently $TH_{\mathcal{HML}}(1) = TH_{\mathcal{HML}}(0)$. Then $\mathbf{AFX}_{\{T\}} true$ cannot be expressed into $\mathcal{HML}$.

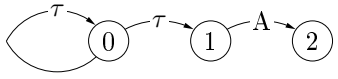


Figure 16: Divergence and inevitablility

Let us consider again the three coffee machines represented 10, all the states of those satisfy the property $\mathbf{AFX}_{\{\text{Tea,Coffee}\}} true$, and in particular states 1, 1' and 1" who correspond to states where a drink was paid and not yet delivered. Let us consider the LTS $\langle Div, d0\rangle$ presented figure 14, none the states of the LTS satisfies $\mathbf{AFX}_{\{\text{Tea,Coffee}\}} true$ and in particular the state $d1$. The fact of having paid drink does not guarantee that one obtains it in a finite time.

The extension of$\mathcal{HML}$ to the atomic propositions (cf 4.3.3), provides us a simple means, in the case of finite LTS, to extend $\mathcal{HML}$ to take into account the concept of divergence.

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ be a LTS, $\mathcal{O} \subset \Sigma$ a subset of observable labels. Let $Div(S)$ be the subset of S defined as follows : $Div(S) =_{Def} \{S \in S : \exists \omega \in \Sigma \setminus \mathcal{O}^{\infty}$ and $s \stackrel{\omega}{\rightarrow}\}$. Thus, for the LTS of the figure 10 we have $Div = \{d1, d2, d3\}$.

One considers $AP_{Div} =_{Def} \{true, Div\}$ and the valuation ν defined by $true \in \nu(S)\ \forall S \in S$ and $Div \in \nu(s) \sigma s \in Div(S)$. By construction logic $\mathcal{HML}(AP_{Div})$ and $\sim_{Div}$, the associated $\pi-$bisimulation, are sensitive to the divergence.

As example we can compare the LTS $\langle M', 0'\rangle$ and $\langle Div, d0\rangle$ presented respectively figures 10 and 14.

$\sim_{Div\ 0} = \{\{d1, d2, d3\}, \{d0, 0', 1', 2', 3'\}\}$
$\sim_{Div\ 1} = \{\{d1, d2, d3\}, \{d0, 0', 2', 3'\}, \{1'\}\}$
$\sim_{Div\ 2} = \{\{d1, d2, d3\}, \{d0\}\{0', 2', 3'\}, \{1'\}\}$
$\sim_{Div\ 3} = \sim_{Div\ 2}$ and consequently $\sim_{Div} = \sim_{Div\ 2}$

The respective initial states of the two LTS $(0, d0)$ are not in the relation of bisimulation ($0 \not\sim_{Div}\ d0$) and consequently $\langle M', 0'\rangle$ and $\langle Div, d0\rangle$ are not "Div"-bisimilar.

4.3.4. *Modal characterizations of other equivalence relations*

One considers the subsets of $\mathcal{HML}$ defined below :

$\mathcal{M} =_{Def} \{F \in \mathcal{HML}, F \text{ does not contain } \wedge\}$ and

$\mathcal{N} =_{Def} \{F \in \mathcal{M}, F \text{ does not contain } \neg\}$

[HM 85] show that which $\mathcal{M}$ is a modal characterization of Co-simulation while $\mathcal{N}$ is a modal characterization of language equivalence.

Strict inclusion between $\mathcal{N}$ and $\mathcal{HML}$ shows the fact that language equivalence is strictly coarser than observational equivalence. Thus, $\mathcal{N}$ do not make it possible any more to express $[A]$ or $false$ which is essential to define properties of deadlock : observational equivalence preserves deadlocks what is not the case for equivalence language. Observational equivalence does not preserve the divergence but can be reinforced to this end [NV 90].

On the other side, the work of [BCG 91] attempts to give a behavioural characterization of logic $\mathcal{CTL}^*$. The relation of equivalence now operates between Kripke's structures presented definition 1. Its presentation is very close to the one of $\sim_N$ equivalences given definition 31.

Definition 56 *Equivalences of Kripke's structures*

Let $\mathcal{KS} = \langle AP, S, \rightarrow, \nu\rangle$ *and* $\mathcal{KS} = \langle AP, S', \rightarrow', \nu'\rangle$ *two Kripke's structures sharing the same set of propositional variables AP*

One defines a sequence of equivalence relations E_{K_0} , E_{K_1} , *...sur* $S \times S'$ *as follows :*

$s\ E_{K_0}\ s'$ *iff* $\nu(s) = \nu'(s')$

$s\ E_{K_{n+1}}\ s'$ *iff*

1. $\nu(s) = \nu'(s')$
2. $\forall s_1 \in S\ : (s \rightarrow s_1) \Rightarrow \exists s'_1 \in S' : s' \rightarrow s'_1$ *and* $s\ E_{K_n}\ s'$
3. $\forall s'_1 \in S' : (s' \rightarrow s'_1) \Rightarrow \exists s_1 \in S : s \rightarrow s_1$ *and* $s_1\ E_{K_n}\ s_1$

Finally, equivalence between Kripke's is defined as follows :
$s\ E_K\ s'$ *iff* $s\ E_{K_i}\ s' : \forall i \geq 0$

The behavioral characterization of logic $\mathcal{CTL}^*$is given by the following property :

Property 57 *behavioral Characterization of* $\mathcal{CTL}^*$
$s\ E_K\ s' \Rightarrow \forall f \in \mathcal{CTL}^*[s \models f \Leftrightarrow s' \models f]$

[BCG 91] also introduced the stuttering equivalence which gives a behavioral characterization of logical temporal $\mathcal{CTL}^*$_X, namely $\mathcal{CTL}^*$without next-time operator.

5. Decidability of the bisimulation and the evaluation of formulas

We now will expose the fundamental results concerning the decidability of the bisimulation of Petri nets and the evaluation of formulas of temporal logic for a Petri net. In chapter 4 of the treating volume of the Petri nets [HAD 01], we saw that the generic properties all were decidables (boundness, accessibility, ...). Moreover the complexity of the checking of certain properties is completely characterized (the boundness property is $\mathcal{EXP}space$-complete) while for others, the problem remains open (accessibility is $\mathcal{EXP}space$-hard but the algorithm of decision is not primitive recursive). As shown by the preceding sections, temporal logic and the bisimulation allows to characterize the behavior of a labelled Kripke's structure in a way finer than through the generic properties. Also one can expect that the procedures of decision are more difficult to obtain. For example, the problem of accessibility is expressed easily in $\mathcal{LTL}$ and $\mathcal{CTL}$.

Example

- in $\mathcal{LTL}$,
  m not accessible from $(R, m_0) \Leftrightarrow (R, m_0) \models$ **G OR** $_{p \in P}\ p \neq m(p)$
- in $\mathcal{CTL}$,
  m nonaccessible since $(R, m_0) \Leftrightarrow (R, m_0) \models$ **AG OR** $_{p \in P}\ p \neq m(p)$

According to the adopted alternative, certain problems are indecidable while, for the decidable alternatives, the majority of the decision methods rely on the decidability of accessibility implying a great complexity consequently. The ways to obtain results of undecidability or of decidability are of very different nature. We will thus follow this cutting in the continuation.

5.1. *Undecidability results*

The usual technique to show that a problem is indÃľcidable consists in reducing another problem indÃľcidable to the initial problem. This technique supposes that one beforehand determined in another manner the indecidability

of a problem. The problem more general than we study is that of the stop of a program.

Theorem 58 (Stop of a program with parameters) *the problem of the stop of a program prog, parameterized by an integer x is undecidable.*

Proof

We will show this result by absurb. Let us suppose that there is program *teststop* with two integers parameters : a representation (by an integer) of a program *prog* and a value of entry of this program. The choice of the representation of the program is here of no importance; for example, one could choose like representation the integer corresponding to the sequence of bits of the program. One will note $\overline{prog}$ this representation. *teststop* returns true if *prog* stops with the provided value and if not returns false. The behavior of *teststop* is unspecified if the first parameter is not the representation of a program.
We build then a program foo with a single parameter which functions thus.

- foo checks that its parameter x is well the representation of a program *prog* (like a compiler does it). If it is not the case, it stops.
- foo calls $teststop(X, X)$. In other words, it tests if the program *prog* stops by taking as entry its representation.
- If $teststop(X, X)$ returns true, then foo runs without end if not it stops.

Let us examine the behavior of $foo(\overline{foo})$.
If $foo(\overline{foo})$ stops $teststopthen(\overline{foo}, \overline{foo})$ returns true and consequently $foo(\overline{foo})$ does not stop what is absurd.
In the contrary case, $teststop(\overline{foo}, \overline{foo})$ returns false and consequently $foo(\overline{foo})$ stops what is absurd. There is not thus program *teststop*. ♢

The fact that the program has a single parameter in entry is not important as indicates it the following corollary. In addition, this one illustrates the principle of reduction.

Corollary 59 (Stop of a program without parameter) *the problem of the stop of a program prog without parameter is undecidable.*

Proof
Let us show that the problem of the stop of a program with a single parameter is reducible with the problem of the stop of a program without parameter. We thus suppose that there is a program *teststopbis* for the problem of the corollary and we describe how to build a program *teststop*. Let *prog* be a program with a parameter and x an integer value. Then *teststop* behaves as follows :

- *teststop* builds the representation of the program $prog'$ without parameter which consists in calling $prog(X)$.
- Then *teststop* calls $teststopbis(\overline{prog'})$ and returns the corresponding result.

Then *teststopbis* cannot exist. ♢

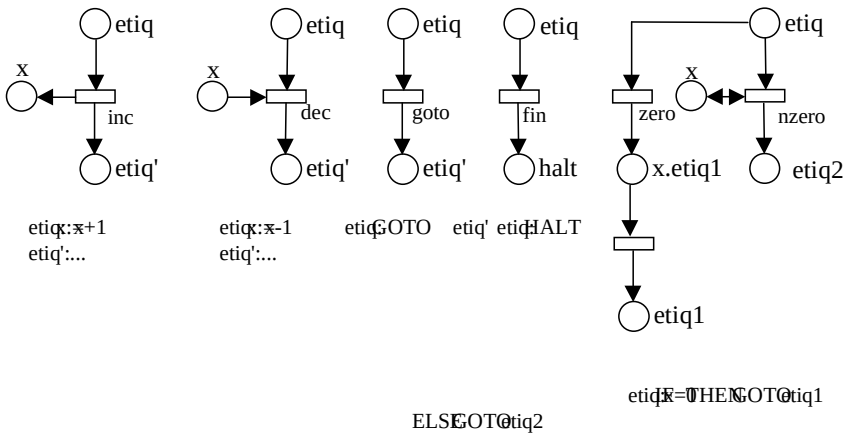


Figure 17: Weak simulation of a program with counter

The choice of the programming language (or the model of computation) is indifferent as from the moment when this one has the minimal constructors conferring to him a expressiveness equivalent to the Turing's machines. To reuse the previous results, one seeks languages adapted to the studied problems. In our case, we will choose the model of the *programs with counter*. The variables of such a program are the counters that are positive integers, initialized by 0. The program is a sequence of instructions ; each instruction is preceded by a label (with the manner of the language *Basic*). The different kinds of instructions are :

- the incrementation $etiq : X := x + 1$
- the decrementation $etiq : X := x - 1$
  When the decrementation is applied to a null counter, it causes an abort of the program **considered as different from the stop**.
- the unconditional jump $etiq$: **GOTO** $etiq'$
- the conditional jump
  $etiq$: **IF** $x = 0$ **THEN GOTO** $etiq1$ **ELSE GOTO** $etiq2$
- the termination $etiq$: **HALT**
  This instruction is necessarily the last instruction of the program.

This program *prog* has only one execution (starting with the first instruction) which can either be infinite, or to abort or to stop when the program reaches the

last instruction. We will propose a weak simulation of a program with counters by a Petri net noted R_{prog} (this name will also apply to the various variants of simulation). One associates with each label, a place which when it is marked indicates that the instruction is the next instruction to be carried out ; initially only the place of the label of the first instruction contains a token. Each counter is translated by a place initially not marked. The translation of the instructions introduces the transitions as indicated in the figure **??** : X. Each transition is labelled by the type of instruction (*inc*, *dec*, *goto*, fin, *zero*, *nzero*). Simulation is weak in the sense where a labelled transition *zero* can be fired although the place x is marked. An exact simulation would require an inhibiting arc from the place x towards the labelled transition *zero*. In other words, among the maximum sequences (finite or infinite), only one of them corresponds to an exact simulation of the program while the other sequences "cheat" by firing in a ill-considered way at least a labelled transition *zero* whereas the corresponding counter is not null. It is then obvious that :

$$prog \text{ terminates}$$
$$\Leftrightarrow$$
$$\text{All maximal runs of } R_{prog} \text{ "cheat" or mark place } halt$$

This leads us to the first results of indecidability.

Theorem 60 (Evaluation of a propositional formula $\mathcal{LTL}$ or $\mathcal{CTL}$) *In a Petri net, the problem of the evaluation of a propositional formula $\mathcal{LTL}$ or $\mathcal{CTL}$ is undecidable.*

Proof
It is enough for us to express the second term of equivalence.
In $\mathcal{LTL}$: **F** ($\mathbf{OR}_{x.etiq \in P}$ $(x.etiq = 1$ **AND** $x > 0)$ **OR** $halt = 1)$
And in $\mathcal{CTL}$: **AF** ($\mathbf{OR}_{x.etiq \in P}$ $(x.etiq = 1$ **AND** $x > 0)$ **OR** $halt = 1)$

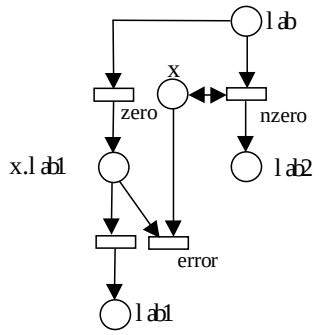
♢

Let us notice that this result is presented within the framework of a semantics on the maximum sequences finite or infinite. One can easily restrict oneself with the infinite sequences by adding a transition who buckles around the place *halt*. This remark is true for the following result.

By modifying the translation of the conditional jump as indicated on the figure 18, we obtain a complementary result.

Theorem 61 (Evaluation of an event-based formula $\mathcal{CTL}$) *In a Petri net, the problem of the evaluation of an event-based formula $\mathcal{CTL}$ is undecidable.*

Proof



lab : IF x=0 THEN GOTO lab1 ELSE GOTO lab2

Figure 18: Another weak simulation of the conditional jump

The equivalence previously mentionned is still valid and it is sufficient to express the second term of equivalence in event-based logic $\mathcal{CTL}$:
$\mathbf{AF}(\mathbf{EX}_{\{erreur\}}\ true\ \mathbf{OR}\ \mathbf{EX}_{\{fin\}}\ true)$ ♢

Let us notice that the operator of branching logic **EX** allows to test the firability what is not possible with an event-based linear logic. We now will transform our weak simulation once again to treat the case of the bisimulation.We add to our network two new "complementary" places y and y' so that that a token either present or in y or in y' but never simultaneously in the two places.

For a marking m of this nature, one will note $\overline{m}$ the marking obtained by reversing the contents of y and y'. One modifies once again the conditional jump but also the last instruction as indicated on the figure 19.

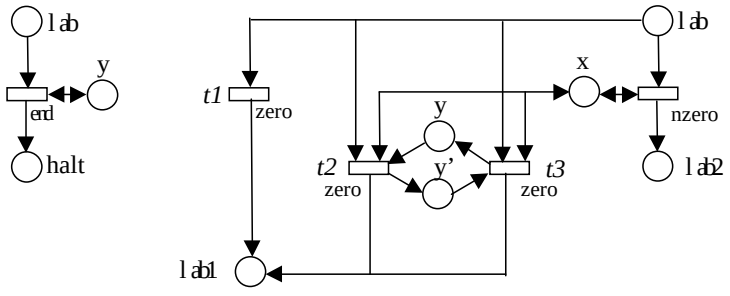


Figure 19: A third weak simulation

Initial marking m_0 is defined by a token in the label of the first instruction and a token in the place y. Here still, one of the maximal sequences of execution corresponds to the simulation of the program with counters. When one "deviates" of exact simulation by firing a labelled transition *zero* whereas the corresponding counter is marked, one can, either swap the contents of y and y' (by t_2 or t_3), or to leave it unchanged (by t_1). The transitions of the t_2 type and t_3 are not used by exact simulation.

Theorem 62 (Bisimulation of Petri nets) *the problem of the bisimulation of two marked nets* (R, m_0) *and* (R', m'_0) *is undecidable.*

Proof
We will show that a program with counters *prog* stops if and only if (R_{prog}, m_0) and $(R_{prog}, \overline{m}_0)$ are not bisimilar

1. *prog* **stops**
Let suppose us that m_0 and $\overline{m}_0$ are not bisimilar. Let $m_0, m_1, \ldots, m_n$ the sequence of markings corrsponding to the exact simulation of *prog*. We show by recurrence that for $i < n$, m_i and $\overline{m}_i$ are bisimilar. Since m_i corresponds to a step of the exact simulation of *prog*, it is possible to speak about the next instruction to execute. If this instruction is an incrementation, a decrementation, an unconditional jumb or the non zero branch of a conditional jump then a single transition labelled with the corresponding action is firable from m_i and $\overline{m}_i$ leading to respectively m_{i+1} and $\overline{m}_{i+1}$. If this instruction corresponds to the zero branch zero of a conditional jump connection, then here also only one transition (t_1) is firable from m_i and $\overline{m}_i$ since the tested (x) counter is not marked; the firing of t_1 leads to m_{i+1} and $\overline{m}_{i+1}$. Let us now examine m_{n-1} and $\overline{m}_{n-1}$. The transition labelled by fin is firable from m_{n-1} but is not firable from $\overline{m}_{n-1}$ since y is not marked. The markings m_{n-1} and $\overline{m}_{n-1}$ are not bisimilar and consequently m_0 are $\overline{m}_0$ are not bisimilar.

2. *prog* **does not stop**
We define the relation $\mathcal{R}$ containing $(m_0, \overline{m}_0)$ and show that $\mathcal{R}$ is a bisimulation. $\mathcal{R} = \{(m, m') \mid m = m'$ where m is a marking reached by the exact simulation of *prog* and $m' = \overline{m}\}$. Of course, it is enough to prove that $\mathcal{R}$ is a bisimulation only for the second type of pair of markings. Let m be a reached marking by the exact simulation of *prog*. Since Puisque *prog* does not stop,the transition labelled by fin is not firable from m. In a case of abort, no more transition is firable both from m and $\overline{m}$. If the next transition to execute is not the zero branch of a conditional jump then a single transition (corresponding to the simulation) is firable from m and $\overline{m}$. This transition corresponds to the

simulation of *prog* and consequently the pair of reached markings belongs to $\mathcal{R}$. If the next transition is a non zero branch then two choices occur from of a conditional jump m (respectively $\overline{m}$), to continue the simulation by firing the transition labelled by *nzero* or to "diverge" from the simulation by firing a transition labelled by *zero* t_1 or t_2 (respectively t_1 or t_3). We show that $\overline{m}$ simulates m (the converse is symetric).

- If transition *nzero* is fired from m, the same is fired from $\overline{m}$ and the pair of reached markings corresponds to the next step of the simulation of *prog*.
- If transition t_1 is fired from m, t_3 is fired from $\overline{m}$ and the reached markings are the same.
- If transition t_2 is fired from m, t_1 is fired from $\overline{m}$ and the reached markings are the same.

$\diamondsuit$

The attentive reader will have noticed that the proof applies to any equivalence - from language equivalence to bisimulation - since if *prog* stop then two nets then are not language equivalent.

In the same way, for the marked nets of the proof, two transitions t and t' are never firable in a concurrent way (i.e $m \geq Pre(T) + Pre(t')$), then the result remains valid for equivalences which take into account concurrent firing.

5.2. *Decidability results*

During this paragraph, we will call upon various concepts introduced in chapter 4 of the volume of the Petri nets [HAD 01] (together semi-linear, technique of shorter sequences, ...). We strongly advise with reader to defer to it for better appreciating what follows.

5.2.1. $\mathcal{LTL}$ *Formulas*

We first study the verification of an event-based temporal logic formula (for instance from linear μ-calcul), formulas which can be represented by an automaton [DAM 92]. Terminal states are interpreted as usual in the case of a finite sequence semantics or as those of a Büchi's automaton when the semantics is expressed in terms of infinite sequence.

The verification procedure for finite systems consists in :

- building the automaton associated with the negation of the formula,

- building the synchronized product between the labelled transition system of the model (i.e. the reachability graph) and the automaton,
- finding a finite sequence (respectively infinite) reaching (respectively a infinite times) a terminal state.

This procedure is obviously not possible in the case of infinite transition systems, but the key of the method that we will expose consists in building a Petri net who generates the labelled transitions system product and then to test in this net the existence of an adequate sequence.

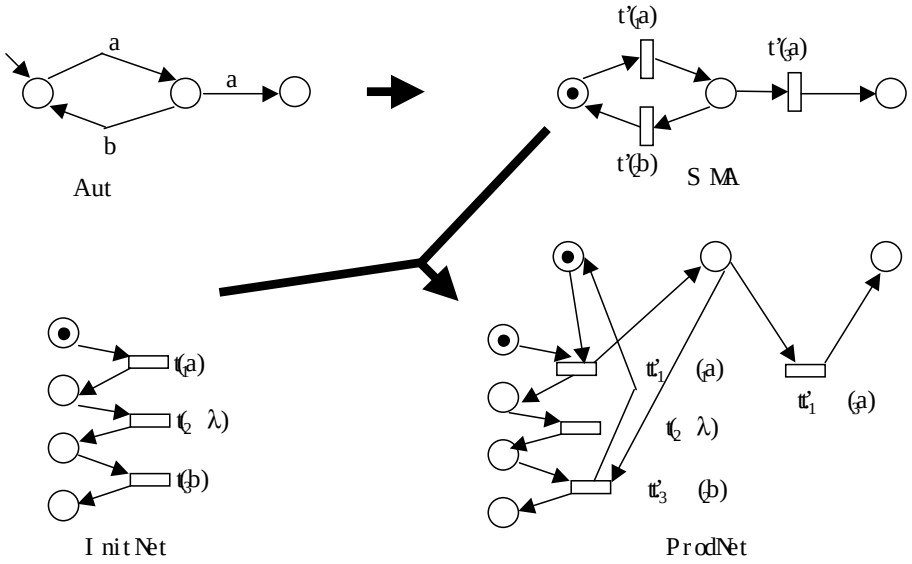


Figure 20: Synchonized product betwen a Petri Net and a automaton

The construction on a Petri Net "product" has been viewed during the chapter treating the study of languages and we recall it briefly here (see figure 20).

- The automaton (`Aut`) is translated in a Petri net (in fact a one-safe state machine) (`SMA`),
- The product net (`ProdNet`) is obtained, from the net associated with the model (`InitNet`) and from the state machine, as follows :
  - The set of places of the product net is defined as the (disjoint) union of the sets of places and the initial marking as the sum of the initial markings,
  - For each pair of transitions sharing the same label we associate a transition ; the input and output arcs of that transition are obtained by union of corresponding arcs in the initial nets. Transition of the initial net labelled by the empty word remain unchanged.

It obviously follows that the observable traces of this net are exactly the

words generated by the initial net and recognized by the automaton (without taking into account the terminal states). This is the departure point of the evaluation method.

Theorem 63 (Evaluation of an event based $\mathcal{LTL}$ formula) *In a Petri net, the evaluation of an event based $\mathcal{LTL}$ formula is a decidable problem (and more generally any formula whose negation is representable as an automaton)*

Proof
This result is valid for any kind of sequences : finite, finite maximal, infinite, divergent. We limit us to the third first kinds since we dont have defined in a precise way a semantics for the divergent sequences.

1. Case of the finite sequences
We search the existence of a finite sequence in the net which marks a place associated with a terminal state of the automaton. In other words, for each of those place, we search to cover the marking defined by the presence of a token in that place. The covering problem has been addressed in and the method of the shorter sequences furnishes a procedure whose complexity is $\mathcal{EXP}space$.

2. Case of the finite maximal sequences
We search the existence of a finite maximal sequence in the product net which marks a place associated with a terminal state of the automaton. Let $Term$ be the subset of that places. In other words, the net has to stop him in a marking where one of the places of $Term$ is marked. The set of these markings is a computable semi-linear set $(\cap_{t\in T}\{m \mid \textbf{NOT } m \geq Pre(t)\} \cap \cup_{p\in Term}\{m \mid m \geq \overrightarrow{p}\}$. We have to know if one of the markings of a semi-linear set is reachable. Since a semi-linear set is a finite union of linear sets, we have successively to test the accessibility of each linear set. Finally, to each linear set $E = \{w \mid \exists\, \lambda_1, \ldots, \lambda_m \; in\mathbb{N}, t.q.\; w = u + \sum_{i=1}^{m} \lambda_i.v_i\}$, we add to the net a transition t_i for i from 1 to m such that $Pré(t_i) = v_i$ and $Post(t_i) = \overrightarrow{0}$. Then, we have to test, in the modified net, if u is reachable; unfortunately, the corresponding algorithm is not recursive primitive.

3. Case of the infinite sequences
We search the existence of an infinite sequence in the product net which marks infinitely often a place associated with a terminal state of the automaton; in other words one of the transitions, having one of the places as an input place, is fired infinitely often. We find again the problem to search an infinite sequence in which a given transition t admits an infinity of occurrences. That is, such a sequence have the form $\sigma = \sigma_1.\sigma_2.\ldots.\sigma_i.\ldots$ where t appears in each σ_i. With the help of extraction lemma of the chapter 3 of [HV 01] applied to the intermediate markings reached by the sequences $\sigma_1.\sigma_2.\ldots.\sigma_i$, we deduce that the existence of such infinite sequence is equivalent to the existence of

a sequence of the form $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$ where $m_1 \leq m_2$ and t having an occurrence in σ_2. Finally, by adding an output place p_t to t, the initial problem is equivalent in this modified net to search a sequence $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$ with $m_1 \leq m_2$ and $m_1(p_t) < m_2(p_t)$. This last problem is also solved using the techniques of the shorter sequences (see [RAC 78, YEN 92] for more details) and leads again to a procedure with a $\mathcal{EXP}space$ complexity. $\diamondsuit$

An interesting question is to known is the decidability is preserved when you consider extensions of Petri Nets. In fact, this evaluation becomes undecidable for almost the totality of the extensions of Petri nets. It is for instance the case for recursive Petri nets and even for restricted models [BOU 96]. However, when you consider only a sequential semantics of the firing of an abstract transition, the problem remains decidable [HAD 00].

5.2.2. *Bisimulation*

We now study the bisimulation of a marked net and a finite transition system. We will use the $\sim_N$-equivalences, introduced by the definition 31, allowing to characterize in the case of finite STE the bisimulation (cf prop 32).

To avoid the ambiguity in our notations, in particular with respect to the origin of the states which we will consider, we will mention the name of the STE explicitly. Thus we will note $\langle \mathcal{LTS}, S\rangle$ to clarify the fact that the state s is a state of the STE $\mathcal{LTS}$.

We introduce two useful notations for the next developments. Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ be a labelled transition system,

- $Inc_n^{\mathcal{LTS}}$ denotes the set of initialized systems incompatible with $\mathcal{LTS}$ for $\sim_n$: $Inc_n^{\mathcal{LTS}} = \{\langle \mathcal{LTS}', s'\rangle \mid \forall s \in S \ \mathbf{NOT}\ \langle \mathcal{LTS}', s'\rangle \sim_n \langle \mathcal{LTS}, s\rangle)\}$
- $\stackrel{*}{\longrightarrow}$ denotes the transitive and reflexive closure of the union of the $\stackrel{a}{\longrightarrow}$. In other words, $\langle \mathcal{LTS}, s\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{LTS}, s'\rangle$ iff s' is reachable from s.

Lemma 64 *Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ be a finite labelled transition system ($n_s = |S|$) and $\mathcal{LTS}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ a labelled transition system, then :* $\forall s \in S,\ \forall s' \in S'$,

$$\langle \mathcal{LTS}, s\rangle \sim \langle \mathcal{LTS}', s'\rangle \Leftrightarrow \begin{cases} \langle \mathcal{LTS}, s\rangle \sim_{n_s} \langle \mathcal{LTS}', s'\rangle \\ \textbf{\textit{AND}} \\ \nexists\ \langle \mathcal{LTS}', s"\rangle \in Inc_{n_s}^{STE}\ s.t.\ \langle \mathcal{LTS}', s'\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{LTS}', s"\rangle \end{cases}$$

Proof
For the implication from left to right, if $\langle \mathcal{LTS}, S\rangle \sim \langle \mathcal{LTS}', \rangle$ then according to property **??** bisimdecis $\langle \mathcal{LTS}, S\rangle \sim_{n_s} \langle \mathcal{LTS}', \rangle$. In addition, by an obvious recurrence on the number of transitions $\stackrel{a}{\longrightarrow}$ which lead of $\langle \mathcal{LTS}', \rangle$ to $\langle \mathcal{LTS}', S"\rangle$

by using the definition of $\sim$, one establishes that there exists s_1 accessible since s (with the same number of transitions) such that $\langle \mathcal{LTS}', S" \rangle \sim \langle \mathcal{LTS}, s_1 \rangle)$ and consequently $\langle \mathcal{LTS}', S" \rangle \sim_{n_s} \langle \mathcal{LTS}, s_1 \rangle$.

For the implication from right to left, we define the relation $\mathcal{R}$ as follows :

$$\langle \mathcal{LTS}, s_1 \rangle \, \mathcal{R} \, \langle \mathcal{LTS}', s'_1 \rangle \; ssi \begin{cases} \langle \mathcal{LTS}, s_1 \rangle \sim_{n_s} \langle \mathcal{LTS}', s'_1 \rangle \\ \textbf{AND} \\ \nexists \, \langle \mathcal{LTS}', s" \rangle \in Inc_{n_s}^{\mathcal{LTS}} \, t.q. \, \langle \mathcal{LTS}', s'_1 \rangle \stackrel{*}{\longrightarrow} \langle \mathcal{LTS}', s" \rangle \end{cases}$$

We show that $\mathcal{R}$ is a relation of bisimulation.
Let us suppose that $\langle \mathcal{LTS}, s_1 \rangle \stackrel{a}{\longrightarrow} \langle \mathcal{LTS}, s_2 \rangle$; then there exists $\langle \mathcal{LTS}', s'_2 \rangle$ such that
$\langle \mathcal{LTS}', s'_1 \rangle \stackrel{a}{\longrightarrow} \langle \mathcal{LTS}', s'_2 \rangle$ and $\langle \mathcal{LTS}, s_2 \rangle \sim_{n_s - 1} \langle \mathcal{LTS}', s'_2 \rangle$.
Since $\langle \mathcal{LTS}', s'_2 \rangle$ is accessible from $\langle \mathcal{LTS}', s'_1 \rangle$, we have :
$\nexists \, \langle \mathcal{LTS}', s" \rangle \in Inc_{n_s}^{\mathcal{LTS}} \, t.q. \, \langle \mathcal{LTS}', s'_2 \rangle \stackrel{*}{\longrightarrow} \langle \mathcal{LTS}', s" \rangle$. In particular,
$\langle \mathcal{LTS}', s'_2 \rangle \notin Inc_{n_s}^{\mathcal{LTS}}$. Thus there exists $\langle \mathcal{LTS}, s_3 \rangle \sim_{n_s} \langle \mathcal{LTS}', s'_2 \rangle$.

By transitivity and the fact that $\sim_n \subset \sim_{n-1}$, $\langle \mathcal{LTS}, s_3 \rangle \sim_{n_s - 1} \langle \mathcal{LTS}, s_2 \rangle$ and under the terms of the property **??** bisimdecis, $\langle \mathcal{LTS}, s_3 \rangle \sim_{n_s} \langle \mathcal{LTS}, s_2 \rangle$.
Again by transitivity, one thus obtains $\langle \mathcal{LTS}, s_2 \rangle \sim_{n_s} \langle \mathcal{LTS}', s'_2 \rangle$.
The case $\langle \mathcal{LTS}', s'_1 \rangle \stackrel{a}{\longrightarrow} \langle \mathcal{LTS}', s'_2 \rangle$ is similar. $\diamondsuit$

This characterization is at the base of the following result.

Theorem 65 (Bisimulation between a net and a finite system) *The problem of the bisimulation beween a marked net $\langle R, m_0 \rangle$, without transition labelled by the empty word, and a finite labelled transition system $\langle \mathcal{LTS}, s_0 \rangle$ is decidable.*

Proof
Like previously $n_s = |S|$. To decide if $\langle R, m_0 \rangle \sim_{n_s} \langle \mathcal{LTS}, s_0 \rangle$, it is enough to verify that :

- for each transition labelled by a firable from m_0 and leading to m_1, there exists s_1 such that $\langle \mathcal{LTS}, s_0 \rangle \stackrel{a}{\longrightarrow} \langle \mathcal{LTS}, s_1 \rangle$ and $\langle \mathcal{LTS}, s_1 \rangle \sim_{n_s - 1} \langle R, m_1 \rangle$,
- Forall s_1 such that $\langle \mathcal{LTS}, s_0 \rangle \stackrel{a}{\longrightarrow} \langle \mathcal{LTS}, s_1 \rangle$, ther exists a transition labelled by a firable from m_0 and leading to m_1 such that
  $\langle \mathcal{LTS}, s_1 \rangle \sim_{n_s - 1} \langle R, m_1 \rangle$.

This obviously leads to a recursive procedure whose depth is limited to n_s.

It remains us to be tested if there is an accessible marking m_1 since m_0 such as $m_1 \in Inc_{n_s}^{\mathcal{LTS}}$. Let us study initially markings belonging to $Inc_{n_s}^{\mathcal{LTS}}$. According to the preceding recursive procedure, to test $\sim_{n_s}$ one examines only firing sequences of length $\geq n_s$. Let us pose v the maximal valuation of an arc of R

and $B = v.n_s$. Let us take two markings m and m' such as $\forall p \in P,\ m(p) \neq me(p) \Rightarrow m(p) \geq B$ **AND** $me(p) \geq B$. These two markings are equivalent for $\sim_{n_s}$. That is to say thus a marking m bounded by B, let us pose $Sup^B(m) = \{me \,|\, me \geq m$ **AND** $\forall p \in P,\ m(p)\}$.

Obviously all the markings in $Sup^B(m)$ are equivalent for $\sim_{n_s}$. We have to note that a marking necessarily belongs to $Sup^B(m)$ for a m bounded by B.

The decision procedure operates in two steps First for each marking m bounded by B, she tests - using the previous procedure - if $m \in Inc_{n_s}^{\mathcal{LTS}}$. Then for each of these m, it search if there exists $m' \in Sup^B(m)$ with m' reachable from m_0. This research consists in testing the accessibility of m in a net augmented with a transition t_p for each place p such that $m(p) = B$, transition defined by $Pre(t_p) = \overrightarrow{p}$ and $Post(t_p) = \overrightarrow{0}$. $\diamondsuit$

In [JAN 99], one will be able to find results more techniques like the existence test of a finite labelled transition system in bisimulation with a Petri net.

Références

[ARN 92] A. Arnold Systèmes de transitions finis et sémantique des processus communicants Masson Paris, 1992.

[AHO 74] Alfred V. Aho, John E. Hopcroft et J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.

[BCG 91] M.C. Browne, E.M. Clarke, O. Grumberg Characterizing finite Kripke structures in propositional temporal logic *Theoretical Computer Science*, 59 :115-131, 1988

[BOU 96] A. Bouajjani et Habermehl P. Constraint properties, semi-linear systems and Petri nets. In *CONCUR'96*, volume 1119 of *LNCS*, 1996.

[BRI 88] E. Brinksma A theory for the derivation of tests In *PSTV'88*, Elsevier Science Publishers B.V., North Holland, 1988

[BRY 86] R. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

[BUC 62] J.R. Buchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, USA, 1962. Stanford University Press.

[CLE 89] R. Cleaveland. Testing equivalence as a bisimulation equivalence. In *Proc. of CAV '89*, volume 407 of *LNCS*, pages 24–37. Springer-Verlag, 1989.

[COU 99] J-M. Couvreur. On-the-fly verification of linear temporal logic. In *Proc. of the Formal Methods'99*, volume 1708 of *LNCS*, pages 253–271. Springer-Verlag, 1999.

[DAM 92] M. Dam. Fixed points of Büchi automata. In *Foundations of Software Technology and Theoretical Computer Science, 12th Conference*, volume 652 of *LNCS*, pages 39–50, New Delhi, India, Décembre 1992.

[DE 87] R. De Nicola Extensional Equivalences for Transitions Systems *Acta Informatica* 24 1987

[DIA 01] M. Diaz Les Réseaux de Petri. Modèles fondamentaux Hermes Science, Traité IC2 Information-Commande-Communication, NřISBN 2-7462-0250-6, 2001, 384p.

[NV 90] R. De Nicola, F. Vaandrager Three Logics for branching bissimulation In *Proc of 5th IEEE Symp. on Logic in Computer Science, 1990*

[DI 86] A. Dicky An algebraic and algorithmic method for analyzing transition s ystems *Theoretical Computer Science*, Vol n^o 46, 1986

[DRI 92] K. Drira Transformation et composition des graphes de refus : analyse de la testabilité Thèse de l'Université P. Sabatier, Toulouse 1992 Rapport LAAS : 92435

[EC 81] Emerson, E. A. and Clarke, E. M. Characterizing Correctness Properties of Parallel Programs as Fixpoints In *Proc. of ICALP'81*, volume 85 of *LNCS*, Springer-Verlag, 1981.

[EC 82] Emerson, E. A. and Clarke, E. M. Using Branching Time Temporal Logic to Synthesize Synchronisation Skeletons Science of Computer Programming, volume 2, pages 241–266, 1982

[EH 85] Emerson, E. A. and Halpern, J. Y. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time Journal of Computer and System Sciences, volume 30 :1, pages 1–24, 1985.

[EME 96] E.A. Emerson. Automated temporal reasonning about reactive systems. In *Logics for Concurrency : Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer Verlag, 1996.

[ESP 97] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34 :85–107, 1997.

[ESP 98] J. Esparza. Decidability and complexity of Petri net problems - an introduction. In *Lectures on Petri Nets I : Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer Verlag, 1998.

[FER 89] J. C Fernandez An Implementation of an efficient algorithm for bissimulation e quivalence *Science Computer Programming*, 13 :219-236, 1989

[GER 95] R. Gerth, D. Peled, M.Y. Vardi et P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15^{th} Workshop on Protocol Specification, Testing and Verification (PSTV'95)*, Varsovie, Pologne, Juin 1995. North-Holland.

[GOD 93] P. Godefroid et G.J. Holzmann. On the verification of temporal properties. In *Proc. 13^{th} Workshop on Protocol Specification, Testing and Verification (PSTV'93)*, pages 109–124, Liege, Belgique, Mai 1993.

[HAD 00] S. Haddad et D. Poitrenaud. A model checking decision procedure for sequential recursive Petri nets. Technical Report 2000/024, LIP6 - Université P. et M. Curie, septembre 2000.

[HM 85] M. Hennessy, R. Milner Algebraic Laws for Nondeterminism and Concurrency *Journal of the A.C.M*, Volume 32(1) :137–161, 1985

[HEN 85] M. Hennessy Acceptance trees *Journal of the A.C.M*, Volume 32(4) :896–928, 1985

[HV 01] S.Haddad, F.Vernadat Méthodes d'analyse des réseaux de Petri In Les Réseaux de Petri. Modèles fondamentaux, Hermes Science, Traité IC2 Information-Commande-Communication, ISBN 2-7462-0250-6, 2001, Chapitre 3, pp.69-117

[HAD 01] S.Haddad Décidabilité et complexité de problèmes de réseaux de Petri In Les Réseaux de Petri. Modèles fondamentaux, Hermes Science, Traité IC2 Information-Commande-Communication, ISBN 2-7462-0250-6, 2001, Chapitre 3, pp.69-117

[JAN 95] P. Jančar. Undecidability of bisimilarity for **P**etri nets and some related problems. *Theoretical Computer Science*, 148 :281–301, 1995.

[JAN 99] P. Jančar, J. Esparza et F. Moller. Petri nets and regular processes. *Journal of Computer and System Sciences*, 59(3) :476–503, 1999.

[LED 90] G. Leduc. Comformance relation, associated equivalence and new canonical tester in Lotos In *PSTV'91*, Elsevier Science Publishers B.V., North Holland, 1991

[MAY 84] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13 :441–460, 1984.

[MIL 89] R. Milner *Communication and Concurrency* Prentice Hall, 1989

[OH 86] E.R. Olderog, C.A. Hoare Specification-Oriented Semantics for Communicating Processes *Acta Informatica* 23 :9–66 , 1986,

[PARK 81] D. Park Concurrency and automata on infinite sequences 5th Conf. On theoretical Computer Sciences pages 167-183 volume 104 of *LNCS*, pages 167–183. Springer Verlag, 1981

[PT 87] R. Paige, R.E. Tarjan Three partition refinement algorithms *SIAM J. Comput*, 1987

[RAC 78] C. Rackoff. The covering and boudedness problems for vector addition systems. *Theoretical Computer Science*, 6(2) :223–231, 1978.

[GLA 90] Rob J. van Glabbeek The Linear Time-Branching Time Spectrum. CONCUR 1990 : 278-297

[VAR 96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency : Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266. Springer Verlag, 1996.

[WOL 83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2) :72–93, 1983.

[YEN 92] H-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96 :119–137, 1992.