

Verification by means of observational equivalence on automata

Didier Vergamini

▶ **To cite this version:**

Didier Vergamini. Verification by means of observational equivalence on automata. [Research Report] RR-0501, INRIA. 1986, pp.17. inria-00076053

HAL Id: inria-00076053
https://inria.hal.science/inria-00076053

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tel (3) 954 90 20

Rapports de Recherche

N° 501

VERIFICATION BY MEANS OF OBSERVATIONAL EQUIVALENCE ON AUTOMATA

Didier VERGAMINI

Mars 1986

Verification by means of observational equivalence on automata

Didier Vergamini
ENSMP - INRIA
Route des Lucioles
Sophia Antipolis
06565 Valbonne Cedex (France)

Abstract

The modelisation and the verification of distributed systems lead us to study finite transition systems or automata as terms of a process algebra supplied with operational semantics. We are interested in the notion of abstract actions on automata and in the equivalence of two automata modulo a set of abstract actions (an observation criterion). We present algorithms allowing to compute automata from terms and some reductions computing a minimal representant of an automaton modulo a criterion. These algorithm are implemented in Lelisp on a Sm90 workstation and their relative speeds are compared from the study of an example: the verification of the alternating bit protocol.

Résumé

La modélisation et la vérification des systèmes distribués nous amènent à étudier des systèmes de transitions finis ou bien automates comme des termes d'une algèbre de processus munie d'une sémantique operationelle. On s'intéresse à la notion d'actions abstraites sur les automates et à l'équivalence de deux automates modulo un ensemble d'actions abstraites (un critère d'observation). On présente des algorithmes permettant de calculer les automates à partir de termes et certaines réductions calculant un représentant minimal d'un automate modulo un critère. Ces algorithmes ont été implantés en Lelisp sur une Sm90 et leurs performances respectives sont comparées à partir de l'étude d'un exemple: la vérification du protocole du bit alterné.

PAPIER RECUPERE ET RECYCLE

0. Introduction

This paper is an illustration of the "proof by computation" method applied to the modelisation and the verification of distributed systems. As an application, we treat an example of communication protocol: a version of the well-known alternating bit protocol [Boch1,Au] following the lines of [Bo2]. It is now usual to consider such systems as the composition of several communicating components acting asynchronously. The main idea is that such entities -also called processes- are characterized by the history of their elementary actions, here considered as non-interruptible, temporally and spatially atomic actions. The model we choose to describe them naturally is the one of **transition systems**. We give **syntax** to describe processes and we give rules to specify the behaviours of syntactic terms, using G. Plotkin's operational semantics [Pl]. An algebra of terms with its operational semantics is called a **processes algebra**. *Meije* [Au&Bo,Au] and *SCCS* [Mi1] are both instances of calculi built on this model. Global systems built on this model are generally becoming very large because of the asynchronicity, so they often are indecipherable. In our example, with four small basic components of 6,3,3 and 6 states, we obtain a global system of 112 states and 668 transitions.

As in [Bo&Su], we distinguish two classes of verification methods:

* temporal logic and verification of properties on global systems [Hail,EMC],
* algebraic theory [Be&Kl,He&Mi,Mi2] and reductions [Li,Be&Te,Si,Na].

In this paper, we follow the latter method. We only deal with **finite** transition systems which are already abstractions of real systems. For instance, in the alternating bit protocol we do not consider the value of the transmitted message but only the proper transmission of a message.

Validation is treated as an **equivalence** between the protocol and the service it is supposed to represent. The equivalence notion is related to the notion of observational criterion representing the sequences that are meaningful in the global system activity. Typically, we don't want to distinguish the time lost inside the system. The equivalence studied in this paper is Milner's observational equivalence [Mi0,Mi1], as in [Na].

The implementation of the verification can be done by mean of equational theories [Be&Kl,Mi2] or by algorithms on automata. We have adopted the latter method, sometimes guided by the former one; the presented algorithms are written in *LeLisp* under the *Ecrins* system [Ma&Ve], which is a metatool for manipulation of process algebras. We evaluate the time of computation on a Sm90 work-station. Several possibilities of strategies of verification are designed and experimented and their relative speeds are compared, a game one cannot easily play within theory.

1. Automata

1.1. Notation

A transition system is a quadruple $\Theta = (Q, s, T, \mathcal{A})$ where

$$\begin{cases} Q & \text{is a non-empty set representing the states of } \Theta \\ T \subseteq Q \times \mathcal{A} \times Q & \text{is the transition set} \\ s \in Q & \text{is the start state} \\ \mathcal{A} & \text{is the set of the actions} \end{cases}$$

We note $p \xrightarrow[T]{a} p'$ for $(p, a, p') \in T$ and $p \xrightarrow{a} p'$ when there is no ambiguity on T.

Automata are transition systems such that Q and T are finite. We won't consider *final states* in our automata because we don't study neither their trace behaviour nor their proper termination.

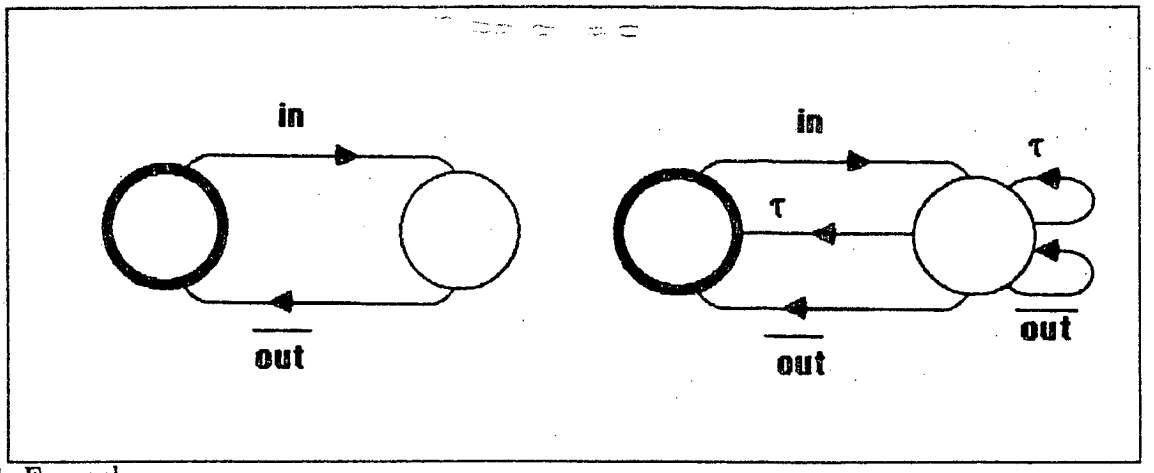


Fig. 1: Example

The figure 1 shows two examples of automata defined as follow:

$$\begin{cases} Q = \{x_1, x_2\} \\ s = x_1 \\ T = \{(x_1, in, x_2), (x_2, \overline{out}, x_1)\} \\ \mathcal{A} = \{in, \overline{out}, \ldots\} \end{cases} \qquad \begin{cases} Q = \{x_1, x_2\} \\ s = x_1 \\ T = \{(x_1, in, x_2), (x_2, \tau, x_1), (x_2, \tau, x_2), \\ \qquad (x_2, \overline{out}, x_1), (x_2, \overline{out}, x_2)\} \\ \mathcal{A} = \{in, \overline{out}, \tau, \ldots\} \end{cases}$$

The first automaton represents a communication line:

* the start state performs the action of collecting a message (in) and shifts to the second state,
* the second state performs the action of transmitting the message ($\overline{out}$) and shifts to the start state. Thus this is a perfect line.

The second one is also a communication line which can loose, duplicate or delay the message:

* the start state performs the action of collecting a message (in) and shifts to the second state,
* the second state has a non-deterministic behaviour; it can perform $\overline{out}$ or τ (the void action) and shift to itself (duplication or delay), or to the start state (good transmission vs loss).

1.2. Reduction of an automaton to its reachable states

When one considers automata with initial state, it is usual to restrict oneself to reachable states: we get here a first reduction.

$\mathcal{A}^*$ stands for the set of words on $\mathcal{A}$, ϵ the empty word and $u;v$ the concatenation of u and v. We extend the transition relation to sequences of actions. We note $p \overset{a}{\underset{T}{\Rightarrow}} q$ after:

$$\begin{cases} p \overset{\epsilon}{\underset{T}{\Rightarrow}} p \\ p \overset{a}{\underset{T}{\rightarrow}} q \implies p \overset{a}{\underset{T}{\Rightarrow}} q \\ p \overset{a;u}{\underset{T}{\Rightarrow}} p' \iff \exists q \quad p \overset{a}{\underset{T}{\Rightarrow}} q \quad and \quad q \overset{u}{\underset{T}{\Rightarrow}} p' \end{cases}$$

This notation is extended to sets of words:

$$for\ U \subseteq \mathcal{A}^* \quad p \overset{U}{\underset{T}{\Rightarrow}} q \iff \exists u \in U \quad p \overset{u}{\underset{T}{\Rightarrow}} q$$

We call such sets **abstract actions** because all actions of the same abstract action are *observed* as equivalent at some degree of abstraction.

In the second transmission line of Figure 1, we have:

$$if \begin{cases} U = \{in; \overline{out}\ ,\ in; \overline{out}; \overline{out}\} \\ V = r* \end{cases} then \begin{cases} x_1 \overset{U}{\Rightarrow} x_1 \\ x_1 \overset{U}{\Rightarrow} x_2 \\ x_2 \overset{V}{\Rightarrow} x_2 \\ x_1 \overset{V}{\Rightarrow} x_1 \\ x_2 \overset{V}{\Rightarrow} x_1 \end{cases}$$

Let $\Theta = (Q, s, T, \mathcal{A})$, we say that $p \in Q$ is reachable iff

$$\exists u \in \mathcal{A}^* \text{ such that } s \overset{u}{\Rightarrow} p \ \left(i.e.\ s \overset{\mathcal{A}^*}{\Rightarrow} p\right).$$

We use the well known **Depth-First Search** algorithm [Ah] to compute the subset of reachable states of an automaton. We call *Acc* the transformation that reduces the states set of an automaton into this subset:

$$\boxed{Acc((Q, s, T, \mathcal{A})) = (Q_{Acc}, s, T \cap (Q_{Acc} \times \mathcal{A} \times Q_{Acc}), \mathcal{A})}$$

where Q_{Acc} stands for the set of reachable states in Q. This gives us our first algorithm.

1.3. Isomorphic automata

Two automata are isomorphic (we note $(Q, s, T, \mathcal{A}) \equiv (Q', s', T', \mathcal{A})$) when there exits a bijection f between theirs states such that:

$$\begin{cases} f(s) = s' \\ p \overset{a}{\underset{T}{\rightarrow}} q \iff f(p) \overset{a}{\underset{T'}{\rightarrow}} f(q) \end{cases}$$

We don't need an algorithm to compute the existence of an isomorphism between two automata because we use the weaker "behavioural equivalence" instead, for which we give a specialized algorithm to compute it in §3.7.

2. Syntax and Semantics

We build a term algebra from a commutative group -standing for the set of actions- and a set of *syntactic constructors* for processes specified by their behaviour semantics. Thus we set us in the framework of Milner's process calculi theory, like CCS [Mi0], SCCS [Mi1], Meije [Au&Bo].

2.1. The commutative group of actions $\mathcal{A}$

As in [Bo2], we consider a set of signal names Sig and $\mathcal{A}$ the **free commutative group** generated by Sig. The product of two actions means their spatial and temporal co-occurence, so it is naturally commutative and associative.

We shall informally let s stand for the reception of the signal named s and its inverse $\overline{s}$ for its emission. So if a process receives a signal while another emits it at the very same instant, this communication is invisible for all other processes; the resulting action is $s.\overline{s} = r$, the neutral element of $\mathcal{A}$.

We say that a signal s divides an action a when s or $\overline{s}$ is a prime factor of a. For instance, s divide $a = s.t$ but do not divide $b = s.\overline{s}.t = t$. Still s divides $\overline{s}$.

2.2. The syntactical construction of the automata algebra *Aut*

The following constructions for terms are slightly the same as the ones described in [Da&Ko]. Let $\mathcal{X}$ a set of variables, $FV(t)$ and $S(t)$ denotes respectively the free variables in t and the sort of t. Then a term is either:

* $\mathbf{0}$,

$$\begin{cases} FV(\mathbf{0}) = \emptyset \\ S(\mathbf{0}) = \emptyset \end{cases}$$

* a variable : $x \in \mathcal{X}$,

$$\begin{cases} FV(x) = \{x\} \\ S(x) = \emptyset \end{cases}$$

* a finite guarded sum : $\sum a_i : t_i$ where (a_i) is a finite set of actions and (t_i) a finite set of terms,

$$\begin{cases} FV(\sum a_i : t_i) = \bigcup_i FV(t_i) \\ S(\sum a_i : t_i) = \bigcup_i \{a_i\} \cup \bigcup_i S(t_i) \end{cases}$$

* a recursive definition : let rec $\{x_i = t_i\}_i$ in t where (x_i) is a set of variables, t and t_i are terms verifying $FV(t_i) \subset \{x_1, \ldots, x_n\}$,

$$\begin{cases} FV(\text{let rec } \{x_i = t_i\}_i \text{ in } t) = \emptyset \\ S(\text{let rec } \{x_i = t_i\}_i \text{ in } t) = S(t) \cup \bigcup_i S(t_i) \end{cases}$$

* a parallel and asynchronous composition : $t_1 \parallel t_2$, where we impose that $FV(t_1) = FV(t_2) = \emptyset$. Then

$$\begin{cases} FV(t_1 \parallel t_2) = \emptyset \\ S(t_1 \parallel t_2) = S(t_1) \cup S(t_2) \end{cases}$$

* a restriction : $t\backslash s$ where $FV(t) = \emptyset$ and s is a signal name, then

$$\begin{cases} FV(t\backslash s) = \emptyset \\ S(t\backslash s) = S(t) - \{a \in S(t), s \text{ divides } a\} \end{cases}$$

Restriction can be viewed as an imperative internal *rendez-vous* or *direct coupling* mechanism used in [Boch1] for instance. In order to simplify expressions, the operator restriction is extended to set of signals:

$$t\backslash\{s_1, \ldots, s_n\} = t\backslash s_1 \backslash \ldots \backslash s_n$$

* a renaming : $t<\psi>$ where t is a term with $FV(t) = \emptyset$ and ψ is a morphism on $\mathcal{A}$,

$$\begin{cases} FV(t<\psi>) = \emptyset \\ S(t<\psi>) = \psi(S(t)) \end{cases}$$

A morphism ψ on $\mathcal{A}$ is represented by a finite list of couples,

$$\psi = <action_1/signal_1, \ldots, action_n/signal_n>$$

meaning $\psi(signal_i) = action_i$.

2.3. The alterning bit protocol

The alterning bit protocol is a well known example in the protocol verification. Using an unsafe medium type, we build a safe system with two such media, an emitter and a receiver. One adds to all the emissions throught the media a bit of control. We don't deal with values in the message (one can suppose that its correctness is yielded by a lower layer protocol). The overall system is designed as in figure 2.

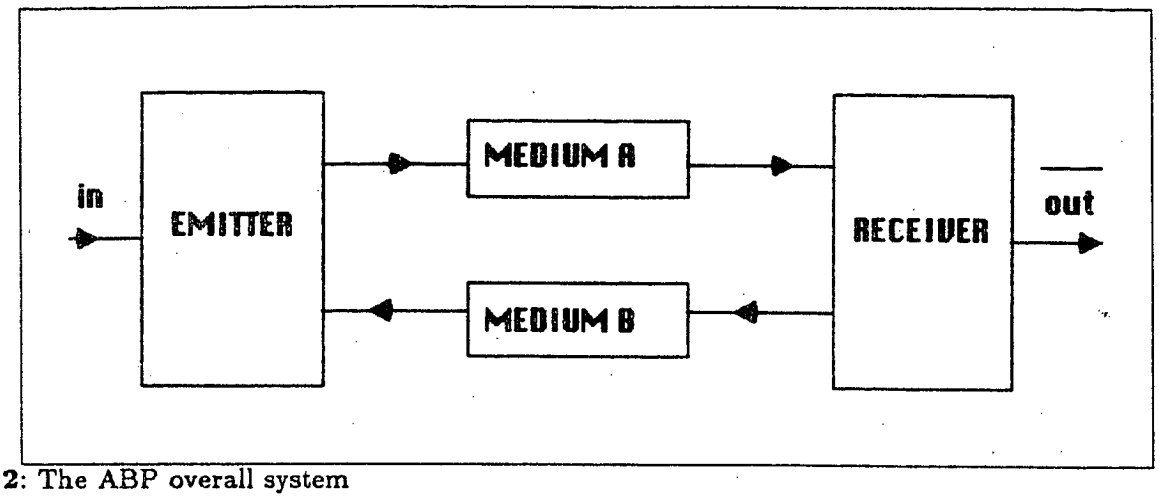


Fig. 2: The ABP overall system

2.3.1. The media

They are ordinary unreliable lines which may either duplicate, lose or delay a message. We suppose that the alterated messages are lost.

The definition of such a medium type is (see fig. 3):

$$MEDIUM = \text{let rec } \left\{ \begin{array}{l} M_0 = in_0 : M_1 + in_1 : M_2 \\ M_1 = \overline{out_0} : M_1 + \overline{out_0} : M_0 + \tau : M_0 + \tau : M_1 \\ M_2 = \overline{out_1} : M_2 + \overline{out_1} : M_0 + \tau : M_0 + \tau : M_2 \end{array} \right\} \text{ in } M_0$$

The first medium is obtained by a renaming:

$$MED_A = MEDIUM < em_0/in_0, em_1/in_1, mr_0/out_0, mr_1/out_1 >$$

The second one is the instantiation:

$$MED_B = MEDIUM < rm_0/in_0, rm_1/in_1, me_0/out_0, me_1/out_1 >$$

The names of signals are chosen so that rm_i is a signal from the receicer to the medium with i as bit of control -and similary for the other ones.

2.3.2. The emitter

The emitter catches a message and transmits it to the medium MED_A and then waits for an acknowledgment. Whenever it receives a acknowledgment with the *unvalid bit* of control, it tries transmission again. It may also decide by itself that its message was lost after a time-out and that it must retransmit it.

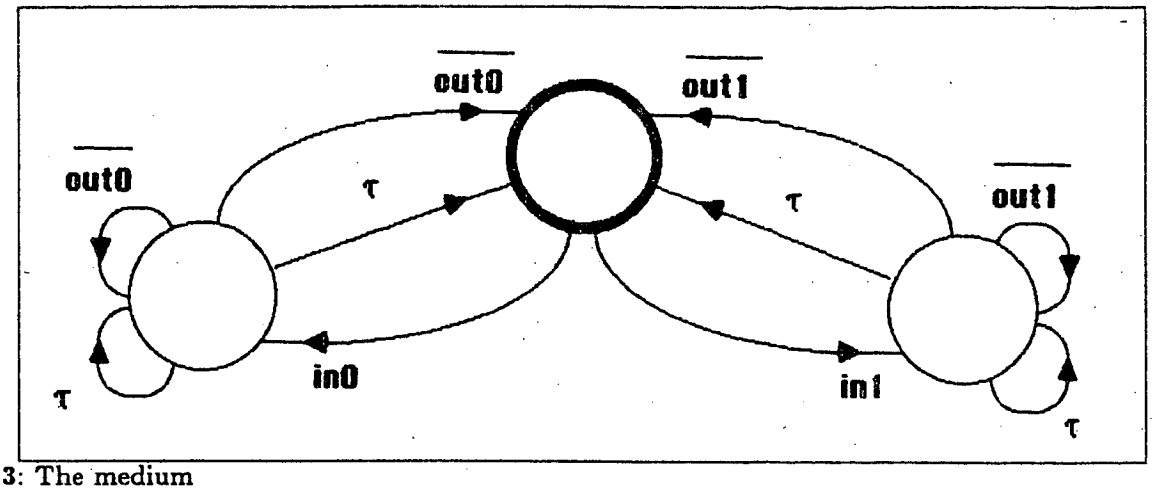


Fig. 3: The medium

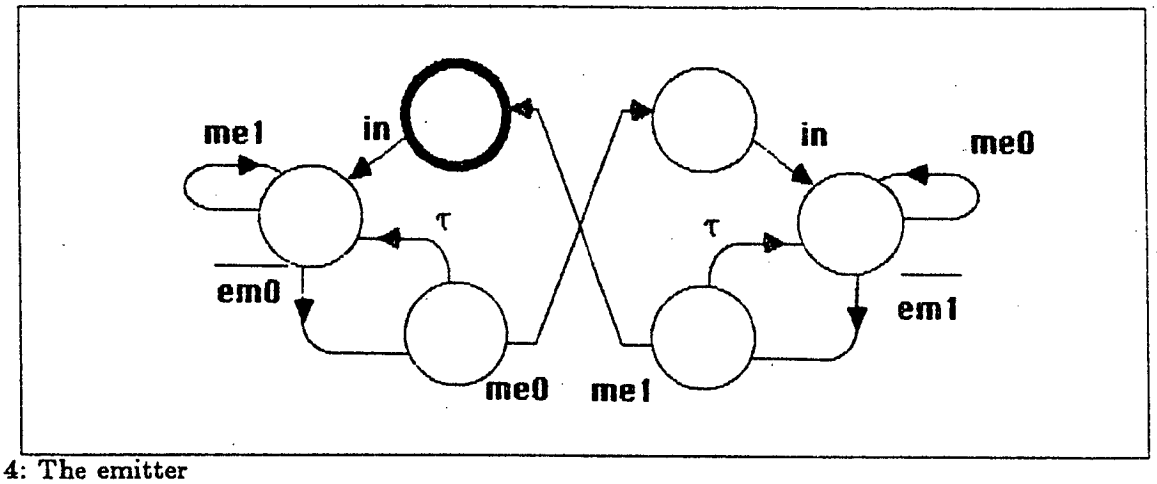


Fig. 4: The emitter

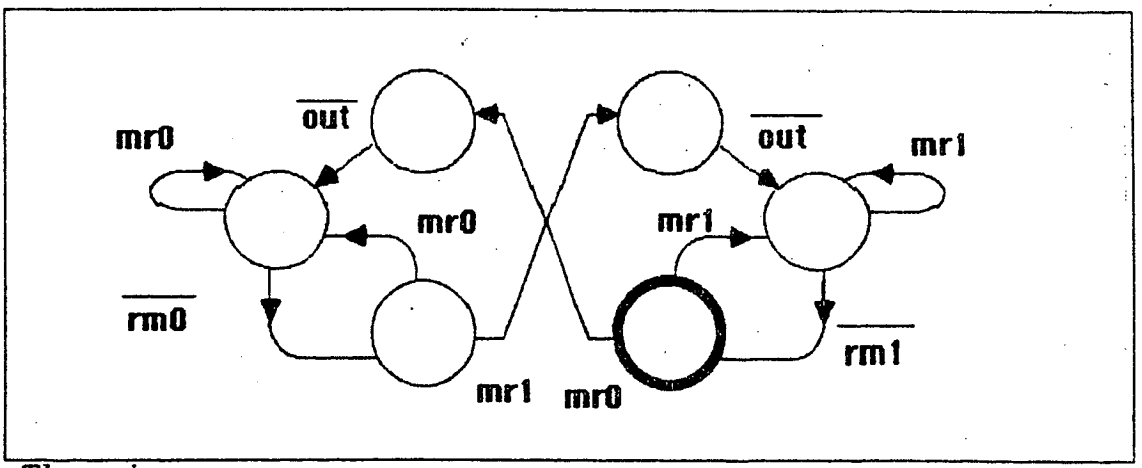


Fig. 5: The receiver

The emitter is represented by the following term:

$$EMITTER = \textbf{let rec} \begin{cases} E_0 = in : E_1 \\ E_1 = me_1 : E_1 + \overline{em_0} : E_2 \\ E_2 = \tau : E_1 + me_0 : E_3 \\ E_3 = in : E_4 \\ E_4 = me_0 : E_4 + \overline{em_1} : E_5 \\ E_5 = \tau : E_4 + me_1 : E_0 \end{cases} \textbf{ in } E_0$$

2.3.3. The receiver

The receiver works the same way as the emitter. When it receives a message with the *valid bit* from the medium, it delivers the message and transmits a corresponding acknowledgment to the medium.

The receiver is represented by the term:

$$RECEIVER = \textbf{let rec} \begin{cases} R_0 = mr_0 : R_1 + mr_1 : R_5 \\ R_1 = \overline{out} : R_2 \\ R_2 = \overline{rm_0} : R_3 + mr_0 : R_2 \\ R_3 = mr_0 : R_2 + mr_1 : R_4 \\ R_4 = \overline{out} : R_5 \\ R_5 = \overline{rm_1} : R_0 + mr_1 : R_5 \end{cases} \textbf{ in } R_0$$

The overall system is the term:

$$ABP = (EMITTER \parallel MED_A \parallel MED_B \parallel RECEIVER) \backslash S$$
$$S = \{em_0, em_1, rm_0, rm_1, me_0, me_1, mr_0, mr_1\}$$

2.4. The behavioural semantics

Each term of the previously defined algebra represents a state of a transition system. Transitions from these terms are given, as usual for CCS, SCCS and Meije by the following rules:

The guarded sum is a non-deterministic operator:

$$\frac{l \in [1..n]}{\sum_{i=1}^{n} a_i : p_i \xrightarrow{a_l} p_l}$$

The rules of the fix point operator are:

$$\frac{t \xrightarrow{a} t'}{\textbf{let rec } \{x_i = t_i\}_i \textbf{ in } t \xrightarrow{a} \textbf{let rec } \{x_i = t_i\}_i \textbf{ in } t'}$$

$$\frac{\textbf{let rec } \{x_i = t_i\}_i \textbf{ in } t[t_i/x_i] \xrightarrow{a} t'}{\textbf{let rec } \{x_i = t_i\}_i \textbf{ in } t \xrightarrow{a} t'}$$

$t[t_i/x_i]$ is the syntactic substitution of the variables x_i by the terms t_i inside of term t.

The parallel composition is asynchronous and non-deterministic:

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad \frac{p \xrightarrow{a} p', q \xrightarrow{b} q'}{p \parallel q \xrightarrow{a.b} p' \parallel q'}$$

The rule of restriction is:

$$\frac{p \xrightarrow{a} p',\ s \text{ does not divide } a}{p\backslash s \xrightarrow{a} p'\backslash s}$$

The rule of renaming is:

$$\frac{p \xrightarrow{a} p'}{p<\psi> \xrightarrow{\psi(a)} p'<\psi>}$$

2.5. Closed terms denote automata

It is easy to see that terms without free variables ($FV(t) = \emptyset$) denote automata. One easily proves this by induction on the term structure. We show here the way to compute this automata following the rules of the behavioural semantics in the two most interesting cases: the parallel composition and the restriction.

* let the automata $\Theta = (Q, s, T, \mathcal{A})$ and $\Theta' = (Q', s', T', \mathcal{A})$ be denoted respectively by t and t'. Then $t \parallel t'$ denotes $\Theta_{t\|t'} = (Q \times Q', (s, s'), T_{t\|t'}, \mathcal{A})$ where

$$(x,y) \xrightarrow[T_{t\|t'}]{a} (x',y') \iff \begin{cases} y = y' & \text{and } x \xrightarrow[T]{a} x' \\ \textit{or} & \\ x = x' & \text{and } y \xrightarrow[T']{a} y' \\ \textit{or} & \\ \exists\, u, v \textit{ such that } a = u.v & \text{and } \begin{cases} x \xrightarrow[T]{u} x' \\ y \xrightarrow[T']{v} y' \end{cases} \end{cases}$$

* let $\Theta = (Q, s, T, \mathcal{A})$ denoting t, $t\backslash s$ denotes the automaton $\Theta' = (Q, s, T', \mathcal{A})$ where

$$x \xrightarrow[T']{a} y \iff x \xrightarrow[T]{a} y \text{ and } s \text{ does not divide } a$$

The restriction operation cuts out some transitions and may leave inaccessible states in Q. So we shall adopt the convention that $t\backslash s$ denotes $Acc(\Theta')$.

The transformation of terms into automata reduced to their reachable states using the behavioural rules is our second algorithm and we note it $\boxed{Aut(t)}$. We give here a first evaluation for the computation of $Aut(ABP)$ in our implementation. The computation time for this automaton is 307 seconds and it has 668 transitions and 112 states: this automaton is of course undicepherable.

The transformation of an automaton into a closed term of our algebra is quite easy and classical: it is the transformation from an automaton to the associated right linear grammar. We note it $\boxed{Term(\Theta)}$.

2.6. Yet a more efficient method to compute overall systems

We present here an algorithm to compute terms like $(t_1 \parallel t_2)\backslash\{s_1, \ldots, s_n\}$. The idea is to compute step by step only the transitions allowed by the restriction operator. Thus we avoid the construction of the overall system $Aut(t_1 \parallel t_2)$.

The method can be described as follows:

* let $T = Q = \emptyset$ and $Reached = \{(s_{t_1}, s_{t_2})\}$
* while $Reached \neq \emptyset$, take an element in $Reached$, put it into Q, compute its transitions with the rules for parallel and restriction operators and put them into T. Then put into $Reached$ any reached state during this step that is not in Q.

* at termination, the result is $(Q, (s_{t_1}, s_{t_2}), T, \mathcal{A})$.

This algorithm for transformation of closed terms into automata is noted $\boxed{Aut'(t)}$.

We can make two other remarks about restriction:

* if s does not divide any element of $S(t_1)$ or any element of $S(t_2)$ then $Aut((t_1 \parallel t_2)\backslash s)$ and $Aut(Term(Aut(t_1\backslash s)) \parallel Term(Aut(t_2\backslash s)))$ are isomorphic,
* if s does not divide any element of $S(t)$ then $Aut(t\backslash s)$ and $Aut(t)$ are isomorphic.

Using these two laws, we can distribute some restrictions and compute:

$$\begin{cases} Half_1 = (EMITTER \parallel MED_B)\backslash me_0\backslash me_1 \\ Half_2 = (MED_A \parallel RECEIVER)\backslash mr_0\backslash mr_1 \\ Aut(ABP) \equiv Aut'(Term(Aut'(Half_1)) \parallel Term(Aut'(Half_2))\backslash em_0\backslash em_1\backslash rm_0\backslash rm_1) \end{cases}$$

Time of computation is now only 24 seconds.

3. Principles of Verification

What is the meaning of **protocol validation ?**

One often considers those properties like *liveness, fairness or deadlock freeness.* An other approach consists in the comparison of a given system and an expected *service* - its specification. This service may be represented among other as *a regular expression* -as in [Boch1]- or as another term of an algebra -as in [Be&Kl bis,Ko]. The comparison will be done for us in terms of equivalence between system and specification. One may use equational reasoning [Be&Kl,Ko] or one may introduce the notions of abstraction criterion and of congruence of automata, with algorithms allowing to compute the equivalences. That is what we are going to do here, using also equational reasoning to justify some strategies in using algorithms.

3.1. Abstraction Criterion and Automaton Abstraction

We call an **abstract action** a subset of $\mathcal{A}^*$. Indeed we are going to consider such subsets of words as higher level actions to abstract the behaviour of automata. We call abstraction criterion any set of abstract actions.

Let C be an abstraction criterion and $\Theta = (Q, s, T, \mathcal{A})$ be an automaton. We denote T_C the transition relation $p \stackrel{U}{\Rightarrow} q, U \in C$. The automaton $\boxed{Abs(\Theta, C) = Acc((Q, s, T_C, C))}$ is the abstraction of Θ on the criterion C.

3.2. Milner's observational criterion

This paper deals only with one particular observational criterion O defined in [Mi1] as

$$O = \{\tau^*\} \cup \bigcup_{a \neq \tau} \{\tau^*; a; \tau^*\}$$

This criterion corresponds to the intention that we do not want to observe the *silent actions*, so we give the abstract action $\tau^* a \tau^*$ the name $\check{a}$ and τ^* is called $\check{\tau}$. It is a particular case of criteria expressing the *freeness from meaning* of some actions.

The computation of T_O is based on the computation of the transitive closure of the relation $\stackrel{\tau}{\rightarrow}$. The algorithm used in our implementation is like the one in [Ah] so that its complexity is $O(n^3)$ where n is the number of states of the automaton.

We call OBS the transformation:

$$\boxed{OBS(\Theta) = Abs(\Theta, O)}$$

The computation of $OBS(ABP)$ give us a 2728 transitions and 112 states automaton in 104 seconds.

3.3. Equipollences induced by criteria

A equipollence over an automaton $\Theta = (Q, s, T, A)$ is a couple $\rho = (C, R)$ where C is a criterion on A and R an equivalence relation over Q verifying

$$\left((p,q) \in R \quad and \quad U \in C \quad and \quad p \overset{U}{\Rightarrow} p' \right) \Longrightarrow \left(\exists q' \quad q \overset{U}{\Rightarrow} q' \quad and \quad (p',q') \in R \right)$$

It is usual to picture this property by the following schema:

$$\begin{array}{ccccc} p & - & R & - & q \\ {\scriptstyle U}\Big\| & & & & \Big\|{\scriptstyle U} \\ \Downarrow & & & & \Downarrow \\ p' & \cdots & R & \cdots & q' \end{array}$$

The family of such relations is a lattice so it has a coarsest element which we note $\sim_C$ and we call C-equipollence the pair $(C, \sim_C)$.

3.4. Construction of the C-equipollence and Quotient of an automaton

Let $\Theta = (Q, T, s, A)$ an automaton and C a criterion. We define $\Theta/\sim_C$ to be the following automaton:

$$\boxed{\Theta/\sim_C = (Q/\sim_C, T_C, [s]_C, C)}$$

where

$$p \overset{U}{\underset{T_C}{\longrightarrow}} p' \iff \exists q, q' \; such \; that \; [q]_C = p, \; [q']_C = p' \; and \; q \overset{U}{\underset{T}{\Rightarrow}} q'$$

We note the reduction of Θ modulo C $\boxed{Red(\Theta, C) = Acc(\Theta/\sim_C)}$.
The computation of $Q/\sim_C$ is done with the following method:

Let R_k be recursively defined by:

$$\begin{cases} \forall p, q \in Q \quad (p,q) \in R_0 \\ (p,q) \in R_{k+1} \iff (p,q) \in R_k \quad \text{and } \forall U \in C \begin{cases} p \overset{U}{\Rightarrow} p' \Longrightarrow \exists q' \quad (p',q') \in R_k \quad and \quad q \overset{U}{\Rightarrow} q' \\ q \overset{U}{\Rightarrow} q' \Longrightarrow \exists p' \quad (p',q') \in R_k \quad and \quad p \overset{U}{\Rightarrow} p' \end{cases} \end{cases}$$

this we picture by the following schema:

$$\begin{array}{ccccc} p & - & R_{k+1} & - & q \\ {\scriptstyle U}\Big\| & & & & \Big\|{\scriptstyle U} \\ \Downarrow & & & & \Downarrow \\ p' & \cdots & R_k & \cdots & q' \end{array}$$

It is easy to show that for finite automata:

$$\boxed{\exists n \leq card(Q) \quad such \; that \quad R_{n+1} = R_n \quad and \quad \sim_C = R_n}$$

This method when applied to ABP gives us the following result:

$$Red(Aut(ABP), C) = \text{let rec } \left\{ \begin{array}{l} x_0 = \breve{r} : x_0 + \breve{in} : x_1 \\ x_1 = \breve{r} : x_1 + \breve{out} : x_0 \end{array} \right\} \; in \; x_0$$

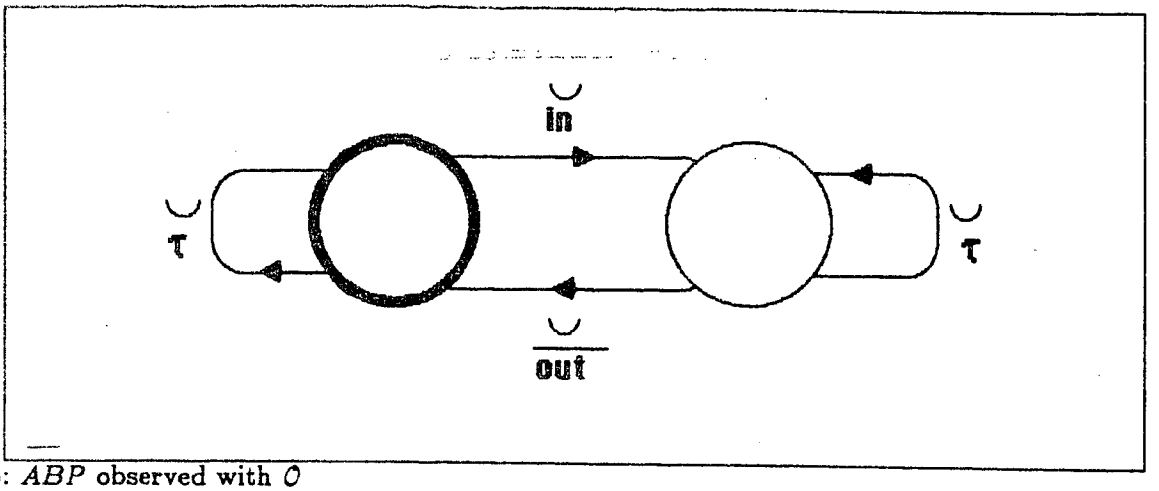


Fig. 6: ABP observed with O

The time of computation is 107 seconds (including the computation of T_O).
We can do two remarks about this result:

* the abstract behaviour of our overall system is easy to analyse,
* it works alike a safe line (with possibility of losing time).

The automaton $Red(\Theta, C)$ is a small-scale model of Θ. So it represents an abstract behaviour. However verification implies comparison of two automata one of which is supposed to be the specification of the other one. For instance, we would like to express that ABP verifies the model of the perfect line given in example in 1.1. So we must introduce the notion of automata equivalence signifying to have the same small-scale model.

3.5. Equivalence of automata

Two automata $\Theta_1 = (Q_1, T_1, s_1, A)$ and $\Theta_2 = (Q_2, T_2, s_2, A)$ are C-equivalent if and only if there exists a relation $R \subseteq Q_1 \times Q_2$ called *bisimulation* by Park [Pa] such that:

$$\begin{cases} (s_1, s_2) \in R \\ \forall (p,q) \in R \quad and \quad \forall U \in C \begin{cases} p \overset{U}{\underset{T_1}{\Rightarrow}} p' \Longrightarrow \exists q' \quad q \overset{U}{\underset{T_2}{\Rightarrow}} q' \quad and \quad (p',q') \in R \\ q \overset{U}{\underset{T_2}{\Rightarrow}} q' \Longrightarrow \exists p' \quad p \overset{U}{\underset{T_1}{\Rightarrow}} p' \quad and \quad (p',q') \in R \end{cases} \end{cases}$$

We denote this relation $\approx_C$. One can show that $\approx_C$ is an equivalence on the automata algebra.

3.6. Properties of $\approx_C$

Property 1

The first result is that we can always limit ourselves to study only the reachable states of an automaton.

$$\boxed{Acc(\Theta) \approx_C \Theta}$$

Property 2

$$\boxed{\Theta \approx_C \Theta' \iff Acc(\Theta/\sim_C) \equiv Acc(\Theta'/\sim_C)}$$

One can make two usefull remarks to prove this:

1 reachability and quotient commute as shown in the diagram:

$$\begin{array}{ccc} & Acc \swarrow \quad \searrow \sim_C & \\ \sim_C \searrow & & \swarrow Acc \end{array}$$

2 two states of an automaton in relation by a bisimulation to a same state of an other automaton are equivalent,

$$(p,q) \in R \text{ and } (p,q') \in R \Longrightarrow q \sim_C q'$$

3.7. An algorithm to compute the C-equivalence of two automata

As we said in §1.3, we shall not need an algorithm to compute the existence of an isomorphism between two automata. We give instead here an other algorithm to directly compute the C-equivalence of two automata $(Q, s, T, \mathcal{A})$ and $(Q', s', T', \mathcal{A})$ where Q and Q' are disjoint.

The main idea of this algorithm is to use the construction by induction of the C-equipollence with the set of states $Q \cup Q'$ and the set of transitions $T \cup T'$. The two automata are C-equivalent if s and s' are in the same equivalence class when the induction stops.

This gives us a new algorithm to compute the C-equivalence of two automata. When applied to ABP and *Perfect-line*, the answer is positive ($Aut(ABP) \approx_O Aut(Perfect\text{-}line)$).

We say that we have performed the validation of ABP with respect to Milner's criterion.

4. Methods of Verification

In this chapter, we expose improvements of the "brute force" methods previously exposed.

4.1. $\approx_O$ is a congruence

We can use the fact that $\mathcal{A}$ and O are in bijection to define a transformation on automata named *obs* which transforms automata on abstract actions into automata on *concrete* actions (belonging to $\mathcal{A}$).

$$\begin{cases} \Theta = (Q, s, T, O) \\ obs(\Theta) = (Q, s, T', \mathcal{A}) \end{cases}$$

where

$$p \xrightarrow[T']{a} q \iff p \xrightarrow[T_O]{\hat{a}} q$$

Then for any automaton Θ on $\mathcal{A}$, we build $Obs(\Theta) = obs(Red(\Theta, O))$ which use the same set of actions. This system is O-equivalent to Θ and can be considered as a canonical representant of the set of automata O-equivalent to Θ:

$$\begin{cases} Obs(\Theta) \approx_O \Theta \\ \Theta \approx_O \Theta' \iff Obs(\Theta) \equiv Obs(\Theta') \end{cases}$$

One easily can verify the following properties:

$$\begin{cases} Obs(Aut(p \parallel q)) \approx_O Aut(Term(Obs(Aut(p))) \parallel Term(Obs(Aut(q)))) \\ Obs(Aut(p \backslash s)) \approx_O Aut(Term(Obs(Aut(p))) \backslash s) \\ Obs(Aut(\sum a_i : p_i)) \approx_O Aut(Term(\sum a_i : Term(Obs(Aut(p_i))))) \end{cases}$$

The third law explains why we chose guarded sum versus ordinary sum. For instance, we have $Aut(\tau : \mathbb{0}) \approx_O Aut(\mathbb{0})$ and $Aut(a : \mathbb{0} + \tau : \mathbb{0}) \not\approx_O Aut(a : \mathbb{0} + \mathbb{0})$ with the classical sum. So we

only deal with the observational congruence called rooted τ-bisimulation in [Be&Kl] and not with the observational equivalence called τ-bisimulation.

These laws allow the modularization of the reduction by applying algebraic laws. In the example of the ABP, we can compute

$$\begin{cases} HALF_1 = Term(Obs(Aut'((EMITTER \parallel MED_B)\backslash\{me_0, me_1\}))) \\ HALF_2 = Term(Obs(Aut'((MED_A \parallel RECEIVER)\backslash\{mr_0, mr_1\}))) \\ Obs(Aut'(ABP)) \approx_O Obs(Aut'((HALF_1 \parallel HALF_2)\backslash\{em_0, em_1, rm_0, rm_1\})) \end{cases}$$

The computation time is 38 seconds versus 131 seconds (24+107) when we have computed $Obs(Aut'(ABP))$.

4.2. Methods of elimination

There are three kinds of eliminations of transitions -and possibly states- described in [Bo2]. By elimination we mean transformation giving a smaller O-equivalent automata.

Elimination of τ-loops

All the transitions $p \xrightarrow{\tau} p$ can be eliminated.

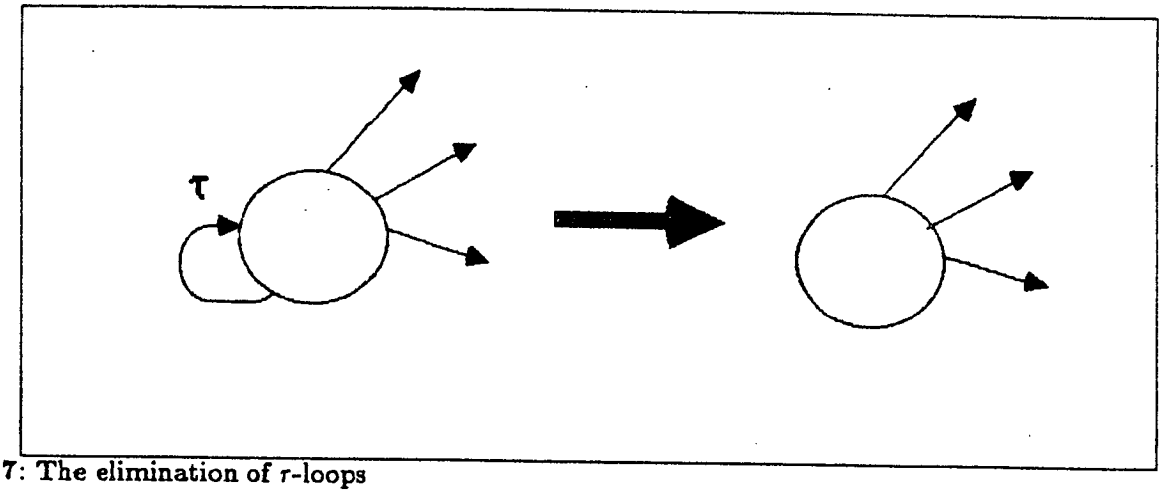


Fig. 7: The elimination of τ-loops

Collapsing τ-transitions

All the states p with only transition $p \xrightarrow{\tau} q$ can be replaced by q (and then we can eliminate p).

Collapsing τ-cycles

All the states which are on a τ-cycle are O-equivalent, so they can be replaced by only one of them with the transitions of all of them.

These three properties are quite easy to prove and enable us to improve the efficiency of our methods of reduction. The most important one is the elimination of τ-cycles which can quickly eliminate a large number of states. On the contrary, the first one is not very useful but shows us that all τ-loops can be implicit.

We call $Elim_\tau^1$ the elimination of τ-cycles and $Elim_\tau^2$ the elimination of τ-transitions. Applying first $Elim_\tau^1$ is slightly faster than applying first $Elim_\tau^2$. We get here a new algorithm:

$$\boxed{Obs'(\Theta) = Obs(Elim_\tau^2(Elim_\tau^1(\Theta)))}$$

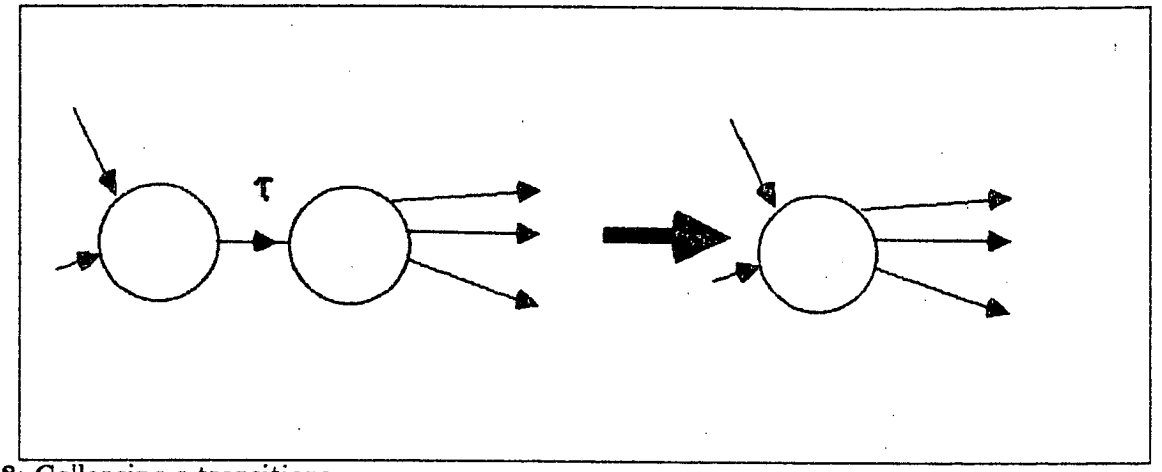


Fig. 8: Collapsing τ-transitions

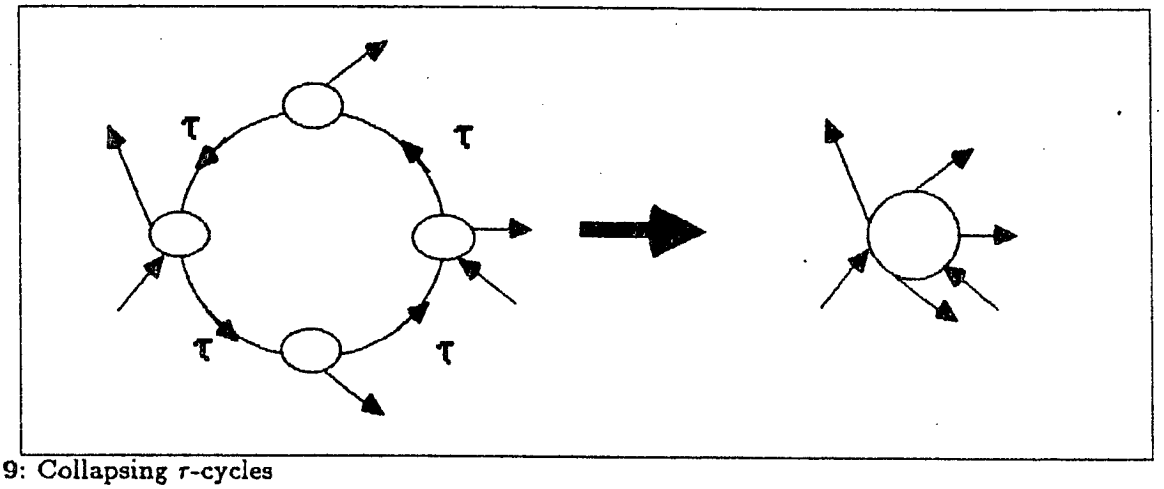


Fig. 9: Collapsing τ-cycles

In the example of the ABP, we can compute

$$\begin{cases} HALF_1' = Term(Obs'(Aut'((EMITTER \parallel MED_B)\backslash\{me_0, me_1\}))) \\ HALF_2' = Term(Obs'(Aut'((MED_A \parallel RECEIVER)\backslash\{mr_0, mr_1\}))) \\ Obs(Aut(ABP)) \approx_O Obs'(Aut'((HALF_1' \parallel HALF_2')\backslash\{em_0, em_1, rm_0, rm_1\})) \end{cases}$$

This construction collapses to 29 seconds versus 38 seconds as our previous best result with $Obs(Aut'(ABP))$.

One can easily see the interest of a good reduction algorithm and of the modularization.

4.3. An example of erroneous protocol

Verification would not be very interesting if we would ever verify but valid systems. The sole reduction can't be viewed as an absolute tool because it can produce an undecipherable automaton. So we need additional tools to make debugging easier. We present here a tool producing a trace of all the transitions leading a particular state of the observed global system.

We use here a wrong version of the alternating bit protocol where we have replaced the emitter

by:

$$EMITTER' = \text{let rec} \begin{cases} E_0 = in : E_1 \\ E_1 = me_1 : E_1 + \overline{em_0} : E_2 \\ E_2 = me_0 : E_3 \\ E_3 = in : E_4 \\ E_4 = me_0 : E_4 + \overline{em_1} : E_5 \\ E_5 = me_1 : E_0 \end{cases} \text{in } E_0$$

The difference between the two versions is the suppression of time-out transition that allows the emitter to decide on reemiting.

$$ABP' = (EMITTER' \parallel MED_A \parallel MED_B \parallel RECEIVER) \backslash S$$
$$S = \{em_0, em_1, rm_0, rm_1, me_0, me_1, mr_0, mr_1\}$$

The overall automaton has 346 transitions and 76 states.
Its observation through O, $obs(Aut(ABP'))$, gives us the following result:

$$\text{let rec} \begin{cases} X_0 = \tau : X_0 + in : X_1 \\ X_1 = \tau : X_1 + \tau : X_2 + \tau : X_3 \\ X_2 = \tau : X_2 + \overline{out} : X_4 \\ X_3 = \tau : X_3 \\ X_4 = \tau : X_4 + \tau : X_0 + \tau : X_3 \end{cases} \text{in } X_0$$

We easily obtain:

$$Aut(ABP') \not\sim_O Aut(Perfect - line)$$

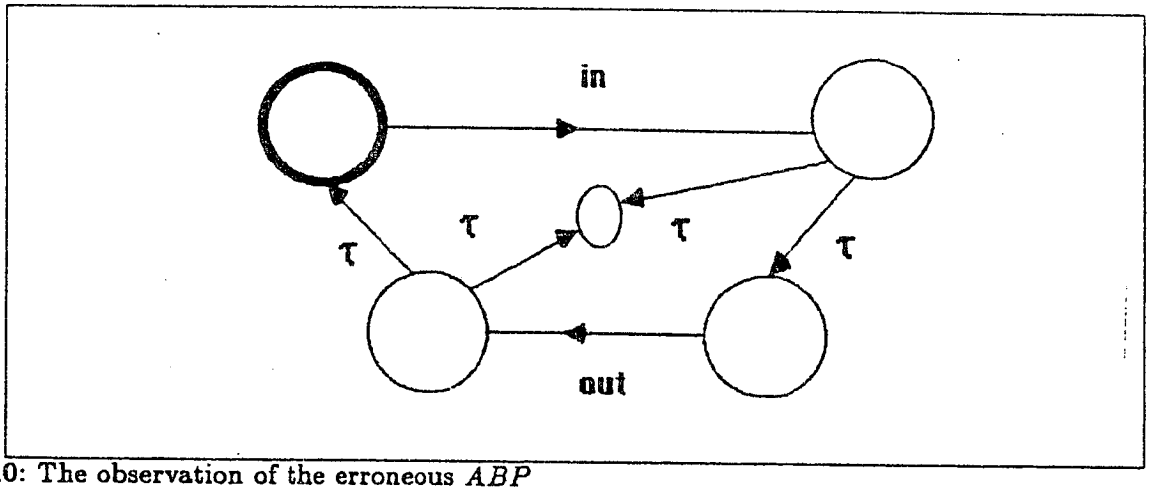


Fig. 10: The observation of the erroneous ABP

The figure 10 shows us the O-equivalent automaton where τ-loops are not pictured.

The question we want to answer is: which sequence of behaviour of each component yields a given state in the observed overall system (for instance, the dead state) ?

To be able to do this, we keep in memory the components of parallel products when we build automata from terms (in Aut or Aut') and the partition giving the equipollent states for $\sim_O$ when we reduce automata by OBS.

In our example, the state X_3 of the result is the equivalence class of the state $[E_2, LR_0, LA_0, R_0]$ in the overall system (we denote $[E_i, LR_j, LA_k, R_l]$ the state of the global system corresponing to the

emitter in the state E_i, the medium MED_A in the state M_k, the medium MED_B in the state M_j and the recceiver in the state R_l). We can find a path in the overall system from $[E_0, LR_0, LA_0, R_0]$ its initial state to this state and decompose the transitions in actions of elementary transitions of each component (we introduce ϵ when a component does not perform any transition).

We obtain the following result:

$$
\begin{array}{r}
[E_0, LR_0, LA_0, R_0] \\
\Big| \; [in, \epsilon, \epsilon, \epsilon] = in \\
\downarrow \\
[E_1, LR_0, LA_0, R_0] \\
\Big| \; [\overline{em_0}, \epsilon, em_0, \epsilon] = \tau \\
\downarrow \\
[E_2, LR_0, LA_1, R_0] \\
\Big| \; [\epsilon, \epsilon, \tau, \epsilon] = \tau \\
\downarrow \\
[E_2, LR_0, LA_0, R_0]
\end{array}
$$

It is easy to analyse that the potential dead-lock is caused by a message lost by the medium while the emitter cannot reemit it, and so still waits for acknowledgement.

5. Conclusion

Processes algebra allows us to use equational approach to improve the study of distributed systems. The combination of algebraic laws with algorithm on automata speeds up the analysis of systems. We summarize the computation times in our example of the alternating bit protocol:

$Obs(Aut(ABP))$	
brute force construction and brute force reduction	307 s.+ 107 s.=414 s.
$Obs(Aut'(Term(Aut'(HALF_1)) \parallel Term(Aut'(HALF_2))\backslash \ldots))$	
fast construction and brute force reduction	24 s. + 107 s.=131 s.
$Obs'(Aut'(Term(Aut'(HALF_1)) \parallel Term(Aut'(HALF_2))\backslash \ldots))$	
fast construction and fast reduction	24 s. + 26 s.=50 s.
$Obs(Aut'(Term(Obs(Aut'(HALF_1))) \parallel Term(Obs(Aut'(HALF_2)))\backslash \ldots))$	
construction using congruence and brute force reduction	38 s.
$Obs'(Aut'(Term(Obs'(Aut'(HALF_1))) \parallel Term(Obs'(Aut'(HALF_2)))\backslash \ldots))$	
construction using fast reduction and congruence	29 s.

This work is to be completed by a study of general criteria and by its application on *real-size examples* such as OSI protocols. A practical tool showing sequences distinguishing two non-equivalent automata is also necessary to make "debug-time" easier.

Acknowledgments

The author would like to thank G. Boudol, E. Madelaine and R. de Simone for their support.

6. Bibliography

[Ah] A. Aho & J. Hopcroft & J. Ullman, "Data structures and algorithms", *Addison-Wesley Publishing Compagny (1983)*

[Au] D. Austry, "Aspects syntaxiques du calcul Meije", *Thèse de troisième cycle, Université Paris 7 (1983)*

[Au&Bo] D. Austry & G. Boudol, "Algèbre de processus et synchronisation", *Theoret. Comput. Sci. 30 p 91-131 (1984)*

[Be&Kl] J.A. Bergstra & J.W. Klop, "A complete inference system for regular processes with silent moves", *Report CS-R8420, Centre for Mathematics and Computer Science, Amsterdam (1984)*

[Be&Kl bis] J.A. Bergstra & J.W. Klop, "Verification of an alterning bit protocol by means of process algebra", *Report CS-R8404, Centre for Mathematics and Computer Science, Amsterdam (1984)*

[Be&Te] G. Berthelot & R. Terrat, "Petri Nets Theory for the Correctness of Protocols", *IEEE TRANSACTIONS ON COMMUNICATIONS, VOL.COM-30, NO.12 (1982)*

[Boch1] G. Bochmann, "Finite State Description of Communication Protocols", *Computer Networks 2 (1978)*

[Bo&Su] G. Bochmann & C. Sunshine, "Formal Methods in Communication Protocol Design", *IEEE TRANSACTIONS ON COMMUNICATIONS, VOL.COM-28, NO. 4 (1980)*

[Bo1] G. Boudol, "Notes on algebraic calculi of processes", *Logigs and Models of Concurrent Systems, NATO ASI Series F13, K. Apt, Ed. (1985)*

[Bo2] G. Boudol, "Calculs de Processus et Vérification", *Rapport Inria 424 (1985)*

[Da&Ko] Ph. Darondeau & L. Kott, "On the observational semantics of fair parallelism", *Lecture Notes in Comput. Sci. 154 p147-159 (1983)*

[EMC] M. Browne, E. Clarke, D. Dill & B. Mishra, "Automatic verification of sequential circuits using temporal logic",*CMU-CS-85-100, Carnegie-Mellon Univ. (1984)*

[Hail] B. T. Hailpern, "Verifying concurrent processes using temporal logic", *Lectures Notes in Comput. Sci. 129 (1982)*

[He&Mi] M. Hennessy & R. Milner, "Algebraic laws for nondeterminism and concurrency", *JACM 32 p137-161 (1985)*

[Ko] C.J. Koomen, "Algebraic specification and verification of communication protocols", *Science of Computer Programming 5 p1-36 (1985)*

[Li] R. Lipton, "Reduction: A Method of Proving Properties of Parallel Programs", *Communications of the ACM Vol. 18, NO. 12 (December 1975)*

[Ma&Ve] E. Madelaine & D. Vergamini, "Ecrins - Manuel d'utilisation", *Rapport INRIA to appear (1986)*

[Mi0] R. Milner, "A Calculus of Communicating Systems", *Lectures Notes in Comput. Sci. 92 (1980)*

[Mi1] R. Milner, "Calculi for synchrony and asynchrony", *Theoret. Comput. Sci. 25 p267-310 (1983)*

[Mi2] R. Milner, "A complete Inference system for a class of Regular Behaviours", *J. of Computer and Systems Sciences 28 p439-466 (1984)*

[Na] E. Najm, "Verification of distributed systems using structured communicating agents nets", *Protocol Specification, Testing and Verification, IV , IFIP 85, p 295-308 (1985)*

[Pa] D. Park, "Concurrency and automata on infinite sequences", *Lecture Notes in Comput. Sci. 104 p167-183 (1981)*

[Pl] G. Plotkin, "A structural approach to operationnal semantics", *Report Daimi FN-19, Comput. Sci. Dept., Aarhus Univ. (1981)*

[Si] J. Sifakis, "Property preserving homomorphisms of transition systems", *Logics of Programs, Lecture Notes in Comput. Sci. 164 p458-473 (1984)*

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique