Acta Informatica 22, 595–627 (1986)

Acta Informatica

A Functional Model for Describing and Reasoning About Time Behaviour of Computing Systems

Paul Caspi[1] and Nicolas Halbwachs[2]

[1] Laboratoire Circuits et Systemes
[2] Laboratoire de Genie Informatique, Institut IMAG, Grenoble, France

Summary. We present a mathematical model of parallel and real time systems behaviour, suitable for problem specification, and for implementation description, analysis and proof. On one hand, this model allows a pure behavioural description mode, and on the other hand it takes into account a metric notion of time. A system is considered as a history transformer, and the history of a variable consists of the sequence of values assigned to it, together with the sequence of times when these assignments take place. A set of tools is proposed, in order to describe such histories, and to perform proofs about them. This model is illustrated by the specification, the description and the proof of a distributed bus arbiter.

Introduction

Generally speaking, the proof of a computing system consists of showing the adequacy of the behaviour of the implemented system to its specified behaviour, this notion of adequacy being defined as an equivalence or preordering relationship over the set of behaviours. Thus, when designing a proof method, the first task consists of formalizing the concept of behaviour.

Classical (sequential and terminating) programs are usually considered as states transformers, and their behaviour is modelled either as input/output predicates, or as functions of states. In order to adapt such approaches to parallel and non terminating programs, which continuously receive their inputs and deliver their outputs, many authors (for instance [2, 9]) have proposed to consider these systems as history transformers. The behaviour of a system is then seen as a set of functional and temporal relations between its input and output sequences.

In the context of parallel programming, many synchronization and communication tools have been proposed, in order to obtain behaviours that are in some sense independent of execution times. This allows to prove the correctness of systems independently of unknown parameters, such as hardware per-

rmances and host computers workload. As a consequence, usual proof meth-ls do not take into account any metric notion of time: two systems are uivalent when they exhibit the same input and output sequences, whatever the time intervals between input and output events. Though these time dependent approaches present many advantages (portability, versatility, easy sign and proof), they cannot apply to systems whose correctness is explicitly ne dependent, that is, mainly, to two important classes of systems:

So called "real time" systems, which can be found in the field of industrial ntrol: On one hand the specification of such systems explicitely contains real ne constraints (response times, sampling frequencies, ...), and on the other nd, in order to meet these time constraints, the implementation of such stems often takes advantage of the knowledge of execution times, so as to ve time consuming synchronization mechanisms.

Hardware systems, which implementation is also often optimized by taking to account the response times of the elementary devices, which can be quite ecisely known.

In this paper, we propose a formalism for the specification and proof of ch real time systems. Earlier drafts of our approach, which is presented in rt 1, can be found in [5, 6] and an extended presentation in [8]. Specific plications to hardware and process control have been presented in [1, 4].

The main idea of our approach is to describe the history of a variable, not ly by means of the sequence of values assigned to it, but also by the quence of times where these assignments take place. Such a sequence of stants will be used to describe what we call an event. Some straightforward athematical tools - precedence relation, time translation, subsequencing and tering - allow then to define temporal relationships between those histories variables. However, this approach quickly exhibits limitations, concerning nsformation and proof capabilities.

These limitations can be, at least partially, eliminated, by considering time quences as increasing functions from integers to time. The theory of Galois presentations then allows to define "pseudoinverses" of these functions. ese are increasing functions, from time to integers, and may be interpreted occurrences counters. This yields many valuable consequences: Many con-uctors for time sequences can also be expressed by means of counters. New nstructors, such as the sum of counters, may be defined. New functions, such variable trajectories, and last and next occurrence time functions, can be tained using functional composition. Finally, the functional composition also ows to express intermittent assertions, and provides a quite powerful func-nal proof technique, the more so as the use of Galois representation proper-s avoids many inductive reasonings.

The second part of the paper will illustrate the use of our formalism on the sign of a distributed bus arbiter, from its initial specifications up to its rdware implementation. Two solutions will be formally described and oved. The former solution is a "traveling token" algorithm which could, to me extend be proved using classical tools, as the response times of the mentary devices do not affect its correctness, but only its overall response ne. On the contrary, the later solution, which is more time effective, behaves

correctly only if the system satisfies some time properties, which can be very precisely defined during the proof.

Part 1. Presentation of the Model

1. Basic Notions

1.1. Time

We always consider a universal time scale, which is that of an external observer to the system. Clearly, the problem, studied by [11], of the relative measuring of time performed by several clocks in a distributed system, is not relevant at the description level.

We shall note $\mathbb{T}$ the set of times. We may, almost always, consider this set as being discrete, and then assimilate it with the set $\mathbb{Z}$ of integers. However, this discretization of time may lead to overspecifications, especially in the field of distributed systems. So we shall present the model under the assumption of continuous time ($\mathbb{T}=\mathbb{R}$). Elements of $\mathbb{T}$ will be called times or dates, when $\mathbb{T}$ will be considered as an affine space, and periods or intervals when the vectorial structure of $\mathbb{T}$ will be considered.

1.2. Events

The basic objects we shall formalize are events and variables. Intuitively, the notion of event covers all that is usually called a state transition in a system or in its environment. So, an assignment to a variable, a clock edge arrival, an execution of a rendez-vous, are instances of events. Now, the same variable may be assigned several times during the life time of a system. We shall consider each of these assignments as an occurrence of the event "assignment to that variable". What we call an event would be considered, in other terminologies [11, 14], as a sequence of events. Our mathematical representation of such events is based upon the following considerations:

- First, at a suitable level of abstraction, we may consider that event occurrences have no duration: they will be considered as cuts in the time line, that separate the past and the future.
- Second, as we are interested in discrete systems, we shall assume that an event has only a finite number of occurrences within a finite period of time.
- Finally, as we aim at handling whole histories, we shall characterize an event e by means of the increasing sequence of its occurrence dates, or, equivalently, by its "time" function, noted $time(e)$. This function applies $\bar{\mathbb{N}}(=\mathbb{N}\cup\{+\infty\})$ to $\bar{\mathbb{T}}(=\mathbb{T}\cup\{-\infty,+\infty\})$, and associates with every natural n, the time of the n-th occurrence of e. The following properties are assumed:

[t1]. $time(e)$ is a loosely increasing function

[t2]. $time(e)(0)=-\infty$

[t3]. $time(e)(+\infty)=\lim_{n\to\infty} time(e)(n)=+\infty$.

omments

The hypothesis [t1] expresses that the occurrences are numbered in chro-ological order. Note that it does not prevent an event from having several multaneous occurrences.

Property [t2] consists of adding to each event a zero-numbered occurrence, hich happens "at the beginning of times". This convention will be convenient, r instance to point at the assignment of the initial value to a variable. loreover, some algebraic justification for it will be given later.

Property [t3] expresses the hypothesis which prevents an event from occur-ig infinitely often during a finite delay. It will also be algebraically justified.

.3. *Variables*

he evolution of a variable during the life of a system will be modelled by eans of the sequence of its values, and by means of an event which occurs ach time the variable is assigned a new value in this sequence. So, a variable x s characterized by a triple $(\mathit{range}(x), \mathit{val}(x), \mathit{asg}(x))$, where:

$\mathit{range}(x)$ is the set of possible values of x;

$\mathit{val}(x)$ is an application from $\mathbb{N}$ to $\mathit{range}(x)$, which value on n is the n-th alue assigned to x;

$\mathit{asg}(x)$ is an event ("assignment to x") so as x takes the value $\mathit{val}(x)(n)$ at he instant $\mathit{time}(\mathit{asg}(x))(n)$.

omments

- $\mathit{val}(x)(0)$ is the initial value of x, taken at time $\mathit{time}(\mathit{asg}(x))(0) = -\infty$;
- Notice that two successive values in the sequence of values of a variable x nay be equal. An occurrence of the event $\mathit{asg}(x)$ does not necessarily cor-espond to an effective change of the value of x.
- As a summary of the precedingly defined concepts, let EV be the set of event ames, VAR be the set of variables names, and VAL the set of values used in escribing a system. Then,

$$\mathit{time} \in \mathrm{EV} \to \mathrm{OCC}$$
$$\mathit{range} \in \mathrm{VAR} \to 2^{\mathrm{VAL}}$$
$$\mathit{asg} \in \mathrm{VAR} \to \mathrm{EV}$$
$$\mathit{val} \in \mathrm{VAR} \to (\mathbb{N} \to \mathrm{VAL})$$

where OCC is the set of increasing functions from $\bar{\mathbb{N}}$ to $\bar{\mathbb{T}}$, satisfying [t1], [t2], [t3].

1.4. *Examples of Application to Time Constraints Expression*

We have now defined the basic objects of our model, and the only properties we assume. Clearly, the time behaviour of any system may be described using these tools, and the whole expressive power of a usual mathematical language. Let us give some examples of such descriptions.

Periodical events. In the field of process control and of signal processing, one must often describe periodical events. Strict periodicity, with period δ, of an event e may be expressed as follows:

$$\forall n \in \mathbb{N}^*,\ \mathit{time}(e)(n+1) = \mathit{time}(e)(n) + \delta$$

where $\mathbb{N}^* = \mathbb{N} - \{0\}$.

However, such a strictly periodical event can be difficult to generate in a computer system. Generally, the notion of periodicity may be interpreted in a loosened sense, such as the following ones:

Maximum Period: two successive occurrences of the event e must be separated by a delay smaller than δ:

$$\forall n \in \mathbb{N}^*,\ \mathit{time}(e)(n+1) \leqq \mathit{time}(e)(n) + \delta.$$

Mean period: e must occur once and only once in each time interval of duration δ, counted from an initial instant t_0:

$$\forall n \in \mathbb{N}^*,\ t_0 + (n-1)\,\delta \leqq \mathit{time}(e)(n) < t_0 + n\delta.$$

Response times. In real time systems specification, when an output variable y is computed as a function of an input variable x ($y = f(x)$, for some function f from $\mathit{range}(x)$ to $\mathit{range}(y)$), a response time, say δ, is frequently required between the acquisition of a value of x and the sending of the corresponding value of y. However, we have encountered such a requirement in contexts where the sampling of x and the sending of y were periodic (for instance in the sense of maximum period defined above) with different periods. In this situation, the response time requirement may be given, at least, three different interpretations:

a) Either the effective renewal periods of x and y must be equal, and their values are in one-one correspondance:

$$\forall n \in \mathbb{N}^*,\ \mathit{val}(y)(n) = f(\mathit{val}(x)(n)) \text{ and}$$
$$\mathit{time}(\mathit{asg}(x))(n) \leqq \mathit{time}(\mathit{asg}(y))(n) \leqq \mathit{time}(\mathit{asg}(x))(n) + \delta.$$

b) Or y may be issued at a lower frequency than x, but each value of y must be computed from a value of x not older than δ:

$$\forall n \in \mathbb{N}^*,\ \exists m \in \mathbb{N}^* \text{ such that}$$
$$\mathit{val}(y)(n) = f(\mathit{val}(x)(m)) \text{ and}$$
$$\mathit{time}(\mathit{asg}(x))(m) \leqq \mathit{time}(\mathit{asg}(y))(n) \leqq \mathit{time}(\mathit{asg}(x))(m) + \delta.$$

c) Or y must be computed at a higher frequency than x, and each new acquisition of x causes the renewal of y within a time period smaller than δ

iis situation mainly occurs when the computation of y involves other vari-
)les than x):

$\forall m \in \mathbb{N}^*, \exists n \in \mathbb{N}^*$ such that
$val(y)(n) = f(val(x)(m))$ and
$time(asg(x))(m) \leqq time(asg(y))(n) \leqq time(asg(x))(m) + \delta$.

These very simple examples show the usefulness of the formalism to remove
me ambiguities due to the use of natural language. However their statement
n be widely simplified thanks to the use of functional notations, which allow,
most cases, to avoid the use of universal quantifiers.

Functional Notations and Applications

1. Notations of Functions

/e shall often use the Church's lambda notation, thus denoting by $\lambda x \cdot f(x)$ the
nction which associates with each x of its domain, the value $f(x)$.

If f and g are functions, respectively from a set S to a set S', and from S' to
', we note $g \circ f$ their composition $\lambda x \cdot g(f(x))$.

Let f and g be two functions from S to S', and $*$ be a binary operator in S',
ien $f * g$ denotes the function $\lambda x \cdot f(x) * g(x)$. In the same way, if R is a binary
lation among the elements of S', R will also denote the corresponding
)intwise relation on functions:

$$f R g \Leftrightarrow \forall x \in S, f(x) R g(x).$$

I will denote the identity function on any set, and if x belongs to S, we
lall also note x the constant function $\lambda y \cdot x$, from any set to S. The context
ill always allow to remove the ambiguities that may result from these over-
mplified notations.

This functional notation allows simpler expressions of systems properties.
or instance, the one to one response time between an input x and an output y
in now be written:

a) $val(y) = f \circ val(x)$

$$time(asg(x)) \leqq time(asg(y)) \leqq time(asg(x)) + \delta.$$

In this example, we have used an operation – the delaying of an event –
nd a relation – the ordering of time functions – which, among others, are of
eneral usage.

2. Precedence Relation

et us note $e \leqq f$ the fact that, for every natural n, the n-th occurrence of e
appens later than the n-th occurrence of f:

$$e \leqq f \Leftrightarrow time(e) \geqq time(f).$$

This precedence relation induces on the set of events a lattice structure, which greatest lower bound and least upper bound operators will be respectively noted *inf* and *sup*. As a matter of fact, since $(\bar{\mathbb{T}}, \leqq, inf, sup, -\infty, +\infty)$ is a complete lattice, then so is $(\bar{\mathbb{N}} \to \bar{\mathbb{T}}, \leqq, inf, sup, -\infty, +\infty)$, where

$$-\infty = \lambda n \cdot -\infty \quad \text{and} \quad +\infty = \lambda n \cdot +\infty.$$

Now, OCC is the subset of $\bar{\mathbb{N}} \to \bar{\mathbb{T}}$ restricted by $[t_1]$, $[t_2]$, $[t_3]$. It is easy to show that the lattice structure is preserved by this restriction, whereas the completeness is lost: for instance $-\infty$ does not belong to OCC, though it can be the limit of a sequence in OCC. However OCC has a maximum element $\top = (-\infty, +\infty, +\infty, \ldots)$. Finally, the *time* function induces the same structure on EV, up to the remark that the order is reversed for common sense reason: an event is smaller when it occurs later, and thus its *time* function is larger. The minimum element of EV will be denoted 0:

$$time(0) = \top.$$

2.3. Delay Operator

If δ is a positive time interval, the effect of the operator D^δ, applied to an event, is to delay every occurrence of e for a time period δ:

$$time(D^\delta e) = time(e) + \delta.$$

The delay operator is increasing and contracting (i.e. $D^\delta e \leqq e$), with respect to $\leqq$.

2.4. Subsequence Relation

A subsequence function is a function r from $\bar{\mathbb{N}}$ to $\bar{\mathbb{N}}$ such that:

- r is increasing,
- $r(0) = 0$ and $r(+\infty) = +\infty$,
- r is strictly increasing on $r^{-1}(\mathbb{N})$.

An event e' is a subevent of an event e if and only if there exists a subsequence function r such that

$$time(e') = time(e) \circ r.$$

This relation will be denoted $e' \subseteq e$. It is an order relation, that induces on EV a lattice structure $(EV, \subseteq, \sqcap, \sqcup)$. This lattice is not complete, but any upper (lower) bounded subset has a least upper bound (resp. a greatest lower bound). $\subseteq$ is coarser than $\leqq$, as

$$e' \subseteq e \Rightarrow e' \leqq e.$$

This relation generalizes to variables: $x' \subseteq x$ iff there exists a subsequence unction r such that

$$val(x') = val(x) \circ r$$
$$time(asg(x')) = time(asg(x)) \circ r.$$

Application. Thanks to these definitions, the two other response times can now ɔe expressed:

b) $\exists y'$ such that

$$val(y') = f \circ val(x)$$
$$asg(x) \geqq asg(y') \geqq D^{\delta}\, asg(x)$$
$$y \subseteq y',$$

c) $\exists y'$ such that

$$val(y') = f \circ val(x)$$
$$asg(x) \geqq asg(y') \geqq D^{\delta}\, asg(x)$$
$$y' \subseteq y.$$

3. Event Counters

Up to this point, the approach presents some limitations, an example of which s the following one: when each occurrence of an output event f requires the ɔrevious occurrence of an input event e, it is easy to write the precedence ɔonstraint $f \leqq e$. But, if each occurrence of f requires the previous occurrence ɛither of the input e or of the input e', the corresponding constraint seems much more difficult to express. A possible solution would be:

$$f \leqq e \sqcup e'$$

but this is not correct when some occurrences of e and e' takes place at the ɜame time. Instead of introducing new constructors on time sequences, we shall ɜhow that this problem finds a very natural solution in terms of event counters, the more so as these counters can be derived from the time functions, thanks to the theory of Galois representations.

3.1. Galois Representations

Let us recall this theory, in the particular case of time functions, in the form of the following theorem. A more general formulation can be found in [15].

Theorem. *Let* $(L, \leqq, \inf, \sup, \bot, \top)$ *and* $(L', \leqq', \inf', \sup', \bot', \top')$ *be two complete lattices, and let* f *be an infinitely* sup*-distributive function from* L *to* L'*, that is a function such that, for any subset* X *of* L*,*

$$\sup'\{f(x) \mid x \in X\} = f(\sup\{x \mid x \in X\}).$$

Then:

- *f is increasing and $f(\bot) = \bot'$.*
- *There exists a unique function f^{-}, from L' to L, such that*

$$f \circ f^{-} \leqq I \quad \text{and} \quad f^{-} \circ f \geqq' I.$$

- $f^{-} = \lambda y \cdot \sup\{x \in L \mid f(x) \leqq' y\}$,
- *f^{-} is infinitely* inf*-distributive.*

The same theorem obviously applies when exchanging $(\inf, \sup, \bot, \leqq)$ by $(\sup, \inf, \top, \geqq)$. The inverse function will then be noted ^{-}f.

3.2. Event Counters

Let e be an event. It is easy to show that the function $time(e)$ is both infinitely inf- and sup-distributive from $\bar{\mathbb{N}}$ to $\bar{\mathbb{T}}$ (this justifies the assumptions on the time functions). Note that, generally speaking, infinite subdistributivity is a stronger property than lattice continuity. However, in the case of totally ordered sets, such as $\bar{\mathbb{N}}$ or $\bar{\mathbb{T}}$, both properties are equivalent.

Then, there exist exactly two functions, $time(e)^{-}$ and $^{-}time(e)$, from $\bar{\mathbb{T}}$ to $\bar{\mathbb{N}}$, such that:

$$time(e) \circ time(e)^{-} \leqq I \text{ and } time(e)^{-} \circ time(e) \geqq I,$$
$$time(e) \circ {}^{-}time(e) \geqq I \text{ and } {}^{-}time(e) \circ time(e) \leqq I.$$

These functions are respectively defined by:

$$time(e)^{-} = \lambda t \cdot \sup\{n \mid time(e)(n) \leqq t\},$$
$$^{-}time(e) = \lambda t \cdot \inf\{n \mid time(e)(n) \geqq t\}$$

from which, we have

$$time(e)^{-}(+\infty) = +\infty \quad \text{and} \quad {}^{-}time(e)(-\infty) = 0.$$

These functions may be given the following interpretation:

Loose Counter. The function $time^{-}(e)$ associates with any time t the rank of the last occurrence of e that has happenned before or at t. We shall rename it as the loose counter of e and denote it by $\ell count(e)$.

Strict Counter. The function $^{-}time(e)$ associates with any time t the rank of the first occurrence that will take place after or at t. At any time but $-\infty$, this can be seen as the rank of the last occurrence strictly before t, plus one. For reasons of practice, we shall prefer to give a special name to the function equal to $^{-}time(e) - 1$, but at $-\infty$. It will be called the strict counter of e, and be denoted by $count(e)$:

$$count(e) = (I - 1) \circ {}^{-}time(e)$$

where $(I - 1) = \lambda n \cdot \max(0, n - 1)$.

Thus:

$count(e)(-\infty)=0$
$count(e)\circ time(e)\leqq(I-1)$
$count(e)\leqq\ell count(e)$.

gure 1 shows the two counters associated with the event e defined by

$$time(e)=\{-\infty,0,1,2,2,4,5,+\infty,\ldots\}.$$

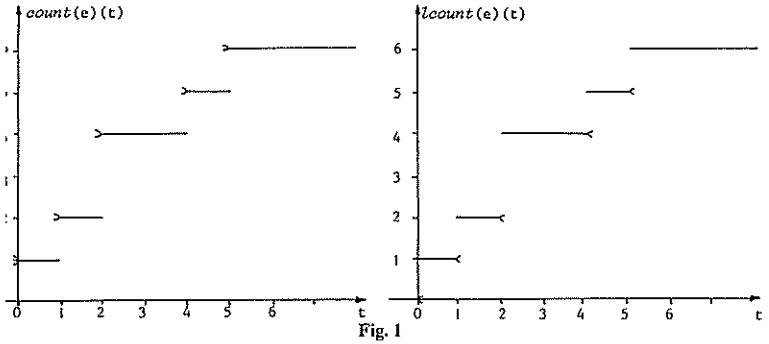

Fig. 1

his example shows that both counters are almost everywhere equal. However, e interest of distinguishing between them comes from their different continuities: *count* is left continuous (sup-distributive) and will be used to express at a property holds just before some considered instants, while *lcount* is ght continuous (inf-distributive) and will be used to express that a property olds just after some considered instants.

me Arithmetics. Some precaution must be taken, when computing with $(I-1)$ te functions, because of the domain limitations to naturals. We have:

$$(I+1)\circ(I-1)=\sup(1,I)\geqq I,\quad (I-1)\circ(I+1)=I.$$

$$(I+1)=(I-1)^{-},\quad (I-1)={}^{-}(I+1).$$

e can also define ${}^{-}(I-1)$ by $\lambda n\cdot$ if $n=0$ then 0 else $n+1$, with

$${}^{-}(I-1)\circ(I-1)\leqq I,\quad (I-1)\circ{}^{-}(I-1)=I.$$

Consequences

t us first restate the precedingly defined constructors:

ose Precedence Relation. As a corollary of the given theorem, we have

$$e'\leqq e\Leftrightarrow time(e)\leqq time(e')$$
$$\Leftrightarrow count(e')\leqq count(e)$$
$$\Leftrightarrow\ell count(e')\leqq\ell count(e).$$

Delay Operator

$$count(D^{\delta}e)=count(e)\circ(I-\delta)$$
$$\ell count(D^{\delta}e)=\ell count(e)\circ(I-\delta).$$

This is due to

$$time(D^{\delta}e)=(I+\delta)\circ time(e)$$
$$(I+\delta)^{-}={}^{-}(I+\delta)=(I-\delta)$$
$$(f\circ g)^{-}=g^{-}\circ f^{-}\ \text{ and }\ {}^{-}(f\circ g)={}^{-}g\circ{}^{-}f,$$

whenever right sides exist.

3.4. Sum of Events

Let LCOUNT be the set of loose counter functions, that is the set of infinitely inf-distributive functions f from $\bar{\mathbb{T}}$ to $\bar{\mathbb{N}}$, such that

$$t<+\infty\Rightarrow f(t)<+\infty.$$

If f is an element of LCOUNT, then ${}^{-}f$, associated to f by Theorem 3.1, is an element of OCC. There is therefore a one-to-one correspondance between LCOUNT and OCC. This provides a new method of defining events, by applying internal operations in the set LCOUNT (the same process applies in the set COUNT of strict counter functions). A typical example is the sum of counters:

LCOUNT is closed with respect to finite summation. Thus we can define the sum $e+f$ of events e and f by

$$\ell count(e+f)=\ell count(e)+\ell count(f)$$

or, equivalently

$$count(e+f)=count(e)+count(f).$$

Intuitively, $e+f$ occurs each time e occurs and each time f occurs. If both e and f occur at the same time, $e+f$ occurs twice at that time. This distinguishes $e+f$ from $e\sqcup f$. It can be shown that

$$e\sqcup f\subseteq e+f$$

and

$$e\subseteq f\Leftrightarrow\exists e'\text{ such that }e+e'=f.$$

We are now able to solve the problem addressed at the beginning of Sect. 3. The expression wanted there is:

$$f\leqq e+e'$$

which is implied by $f\leqq e\sqcup e'$ but the converse is not true.

m and Order. The sum is order preserving with respect to both $\leqq$ and $\subseteq$. or instance,

$$\forall e, f, g, e \leqq f \Rightarrow e+g \leqq f+g.$$

e Sum Structure. 0 is a neutral element for the sum. So (EV, +, 0) is an belian semigroup. We can also define integer events by:

$$time(k) = \lambda n \cdot \text{if } n \leqq k \text{ then } -\infty \text{ else } +\infty.$$

r instance, $\ell count(1)$ is equal to 1 but at $+\infty$, and $count(1)$ is equal to 1 t at $-\infty$.

Note that $^{-}time(e) = count(e+1)$. This allows a concise notation for the ood interleaving" of two events:

$$e \leqq f \leqq e+1$$

ans that, for every n, the n-th occurrence of f succedes to the n-th occurrence e but precedes its $n+1$-th occurrence.

Functional Composition

ent counters allow to build useful functions, using functional composition.

. Current Values of a Variable

x is a variable, one may define two functions giving the value of x at each ne:

$$current(x) = val(x) \circ count(asg(x)),$$
$$\ell current(x) = val(x) \cdot \ell count(asg(x)).$$

mments. This definition of the current value of a variable by means of two nctions may seem surprising. In fact, it avoids to answer the trifling question hat is the value of a variable x at the instants of assignment to x", while owing to denote its values just before ($current(x)$) and just after ($\ell current(x)$) assignments.

. Last and Next Occurrence Functions

e is an event, the functions $\ell ast(e)$ and $next(e)$ respectively give, at each ne t, the time of the last occurrence of e strictly preceding t, and the time of next occurrence strictly following t. These functions may be expressed as llows:

$$\ell ast(e) = time(e) \circ count(e),$$
$$next(e) = time(e) \circ \ell count(e+1).$$

Their loosened counterparts are noted $\ell\ell ast(e)$ and $\ell next(e)$:

$$\ell\ell ast(e) = time(e) \circ \ell count(e),$$
$$\ell next(e) = time(e) \circ count(e+1)$$

with the obvious property:

$$\ell ast(e) \leqq \ell\ell ast(e) \leqq I \leqq \ell next(e) \leqq next(e).$$

4.3. Strict Precedence and Single Occurrence

The loose precedence $e \leqq f$ has been defined equivalently by

$$time(e) \geqq time(f),$$
$$count(e) \leqq count(f),$$
$$\ell count(e) \leqq \ell count(f).$$

If we want to express the stronger property, which will be noted $e < f$, that e occurs always strictly after f, we could write

$$\ell count(e) \leqq count(f)$$

but this implies that $count(f)(+\infty) = +\infty$, which means that f has infinitely many occurrences at finite time. In order to avoid this restriction, we must be able to express that a property holds either up to the last finite occurrence of e (if it exists) or to infinity (otherwise). This is achieved thanks to the following tools:

Definition. For any event e, let us define the function $actual(e)$, from $\bar{\mathbb{N}}$ to $\bar{\mathbb{N}}$ as

$$actual(e) = \inf(I, count(e)(+\infty)).$$

Useful Properties

- $actual(e) \leqq I$,
- $f \leqq e \Rightarrow actual(f) \leqq actual(e)$,
- $f \leqq e \Rightarrow actual(f) \circ actual(e) = actual(e) \circ actual(f) = actual(f)$,
- $actual(e) \circ count(e) = count(e)$,
- $actual(D^{\delta} e) = actual(e)$.

The function $actual(e)$ allows to define the strict precedence of events as follows:

$$e < f \Leftrightarrow actual(e) \circ \ell count(e) \leqq count(f).$$

Properties of Strict Precedence

- $e<f \Rightarrow e \leqq f$

since

$$count(e)=actual(e)\circ count(e) \leqq actual(e)\circ \ell count(e) \leqq count(f)$$

- $e \leqq f<g \leqq h \Rightarrow e<h$

since

$$e \leqq f<g \leqq h \Rightarrow actual(e)\circ \ell count(e) \leqq actual(e)\circ \ell count(f) \leqq actual(f)\circ \ell count(f) \leqq count(g) \leqq count(h)$$

- $e<f \Rightarrow count(e) \leqq count(f)\circ \ell ast(e)$

since $e<f$ implies

$$actual(e) \leqq actual(e)\circ \ell count(e)\circ time(e) \leqq count(f)\circ time(e)$$

and

$$count(e)=actual(e)\circ count(e) \leqq count(f)\circ time(e)\circ count(e).$$

As a general remark, the composition with $\ell ast(e)$ will often be used to express that a property holds on each actual occurrence of e, thus avoiding irrelevant problems "at the end of times".

An event e which always occurs once at a time, except at $+\infty$, is called "single occurrent". One can easily see that this property may be expressed by $e<e+1$, and implies:

$$count(e)=count(e+1)\circ \ell ast(e).$$

5. Conditions, Filtering and Subsequences

5.1. Conditions

A condition is a mapping from $\bar{\mathbb{T}}$ to {true, false}, whose value on $-\infty$ is true. Conditions may be built by bracketting denotations of boolean valued functions of time. For instance:

- if x is a variable such as $range(x)=\{\text{true, false}\}$, then $[current(x)]$ and $[\ell current(x)]$ are conditions;
- if F and G are functions from $\bar{\mathbb{T}}$ to the same set S, and if R is a binary relation over S, $[FRG]$ denotes the condition which is true on t in $\mathbb{T}$ if and only if $F(t)RG(t)$. So, if e and e' are events, if x and x' are variables, then $[count(e) \geqq count(e')]$, $[\ell ast(e) \geqq \ell\ell ast(e')]$, $[current(x)=current(x')]$ are examples of such conditions.

Let *true* denote the always true condition, and *false* denote the condition always false but at $-\infty$. Usual boolean operators (negation, noted $\neg$, disjunction, noted $\vee$, and conjunction, noted $\wedge$) may be applied to conditions. Note that, if C is a condition, both C and $\neg C$ are true at $-\infty$.

Conditions will be mainly used for events "filtering".

5.2. Events Filtering

If e is an event and C is a condition, the event "e filtered by C", noted e/C, is the event which occurs whenever e occurs when C is true. It may be formally defined by

$$e/C=\sqcup\{f \leqq e \mid C\circ \ell ast(f)=true\}.$$

Equivalently, $f=e/C$ if and only if there exists f' such that $e=f+f'$ and

$$C\circ \ell ast(f)=\neg C\circ \ell ast(f')=true.$$

If C and C' are exclusive conditions (i.e. $C \wedge C'=false$), then

$$e/C+e/C'=e/(C \vee C').$$

As a consequence, $e/C+e/\neg C=e$.

The interaction of the filtering operator with other operators we have defined are complex. They may be studied by means of the subsequence functions, which associate with each occurrence of the result of the filtering, a simultaneous occurrence of the filtered event.

5.3. Subsequence Functions

By definition of the subsequence ordering, if $e \leqq f$ there exists a subsequence function r such that $time(e)=time(f)\circ r$. Such a function will be called a subsequence function associated with the couple (e,f). Two subsequence functions r and r' associated with (e,f) are said to be equivalent, if and only if they differ only on numbers of occurrences which occur at infinity, that is

$$r\circ count(e)=r'\circ count(e).$$

The following results have been shown [8]:

a) If $e \leqq f$, the subsequence functions associated with (e,f) are the subsequence functions r such that:

$$count(f+1)\circ time(e) \leqq r \leqq \ell count(f)\circ time(e).$$

b) If $e \subseteq f$, then the functions

$$(count(f) - count(e)) \circ time(e) + I$$

and

$$(\ell count(f) - \ell count(e)) \circ time(e) + I.$$

are respectively the least and the greatest subsequence functions associated with (e,f).

c) A necessary and sufficient condition for e to result from a filtering of f by some condition, is that (e,f) admits only one subsequence function, up to equivalence.

d) If $e = f/C$ and $e' = f'/C'$, and if $C \circ time(f) = C' \circ time(f')$ then the couples (e,f) and (e',f') admit the same subsequence function.

This last result may be used to "project" on e and e' a property satisfied by f and f'. For instance, if (e,f) and (e',f') admit the same subsequence function r, one may derive:

$$\begin{aligned} f \leqq f' &\Rightarrow time(f) \geqq time(f') \\ &\Rightarrow time(e) = time(f) \circ r \geqq time(f') \circ r = time(e') \\ &\Rightarrow e \leqq e'. \end{aligned}$$

In the same way, this result implies:

$$C = C \circ (I + \delta) \Rightarrow D^{\delta}(e/C) = (D^{\delta} e)/C.$$

6. Causality and Induction

Let us explain the matter of this section by an example: Let S be a system in an environment E (see Fig. 2 a). S receives from E an input event e, and sends to E its outputs. Let C be some condition, and assume the following property is to be proved:

P1: "Whenever S receives e, the condition C holds".

Now, consider the system S' (Fig. 2 b) built by adding to S a device for filtering the input e by C, and assume the following property has been shown:

P2: "Whenever S' receives e, the condition C holds".

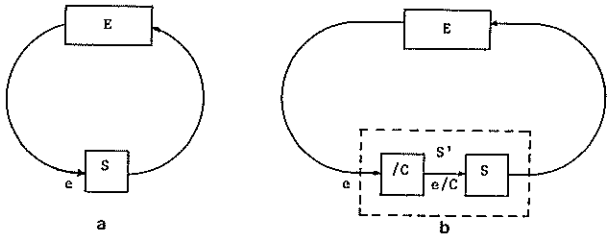

Fig. 2.

The question is: Does this property imply that the filtering is useless, and, so, that the property P1 is proved? More generally, may we replace a subsystem S' by another subsystem S, while S and S' are only equivalent under some assumption concerning the remainder of the whole system (here, the environment E), assumption that may concern the outputs of the subsystem (here, the outputs of S' may affect the elaboration of e by E)? The answer to this question is not so obvious, for the following reasons:

- It needs a proof by induction, which would be, informally, as follows, for the considered example: P2 implies that C holds on the first occurrence of e; if C holds on the n first occurrences of e, then the outputs answered by S and S' to these n first occurrences are the same; so the $(n+1)$th occurrence of e, elaborated by E from these outputs, is the same in both systems; so, from P2, C holds on this $(n+1)$th occurrence.

- This proof makes use of a temporal causality assumption: the same causes have the same consequences, or, more precisely, the behaviour of a system at a given time only depends on its past. But, using our formalism, one may write down specifications which transgress this causality principle.

- Moreover, in the case of continuous time, some care must be taken for the inductive proofs to be valid: An ill-considered reasonning over integers may indeed be erroneous, like the following: Having proved that a property holds on a sequence of intervals $[-\infty, t_n]$, where (t_n) is a strictly increasing sequence of instants, one deduces that the property holds at any time. However, if the sequence (t_n) converges, this proof is only valid up to its limit. In order to avoid this kind of inconsistencies, we shall need a stronger assumption than the single causality. This assumption will be called "overcausality", and will impose that there exists a strictly positive delay ε, such that the behaviour of a system at each time t only depends upon its history until $t - \varepsilon$.

Our point of view, which considers whole histories rather than states, is awkward for inductive reasonning. So, we shall prove, once and for all by induction, a theorem, which allows in many cases to avoid inductive proofs. First, we make our causality assumptions more precise.

6.1. Definitions

Let $\mathbb{F}$ be a set of total functions from $\bar{\mathbb{T}}$ to a set S. If $f \in \mathbb{F}$, its restriction to $[-\infty, t]$ will be noted $f[t]$.

- A mapping G, from $\mathbb{F}$ to $\mathbb{F}$, a binary relation R on $\mathbb{F}$, will be said to be causal, if and only if, for each t in $\bar{\mathbb{T}}$, and for each f and f' in $\mathbb{F}$,

$$\begin{aligned} f[t] = f'[t] &\Rightarrow G(f)[t] = G(f')[t] \\ &\Rightarrow \{f''[t] \mid f R f''\} = \{f''[t] \mid f' R f''\}. \end{aligned}$$

- A mapping G, from $\mathbb{F}$ to $\mathbb{F}$, a binary relation R on $\mathbb{F}$, will be said to be overcausal, if and only if there exists $\varepsilon>0$ such that, for each t in $\overline{\mathbb{T}}$ and for each f and f' in,

$$f[t]=f'[t]\Rightarrow G(f)[t+\varepsilon]=G(f')[t+\varepsilon]$$
$$\Rightarrow\{f''[t+\varepsilon]\,|\,fRf''\}=\{f''[t+\varepsilon]\,|\,f'Rf''\}.$$

5.2. *Properties*

a) if G is a total causal function, if R is an overcausal relation, then the relation $G\cdot R=\{(f,f')\,|\,G(f)\,Rf'\}$ is overcausal.

b) If R is a total ($\mathrm{Dom}(R)=\mathbb{F}$) overcausal relation, and if $f[t]\in\{f'[t]\,|\,fRf'\}$, there exists $\bar{f}$ in $\mathbb{F}$ such that $\bar{f}R\bar{f}$ and $\bar{f}[t]=f[t]$.

Proof. The property (a) is a straightforward consequence of the definitions. The property (b) needs a proof by induction: Let $P(f,t)$ be the property

$$f[t]\in\{f'[t]\,|\,fRf'\}.$$

Then, $P(f,t)$ implies that there exists f_1 such that fRf_1 and $f_1[t]=f[t]$. From the overcausality of R, it implies that

$$\{f'[t+\varepsilon]\,|\,fRf'\}=\{f'[t+\varepsilon]\,|\,f_1Rf'\}.$$

Now, $f_1[t+\varepsilon]$ belongs to the former of these sets, and so it belongs also to the latter. So, we have $P(f_1,t+\varepsilon)$. So, one can build a sequence $f_0=f,f_1,f_2,\ldots,f_n,\ldots$ such that, for any natural n,

$$f_n[t+n\varepsilon]=f_{n+1}[t+n\varepsilon]\quad\text{and}\quad P(f_n,t+n\varepsilon).$$

The sequence $(t+n\varepsilon)$ diverges, so the sequence of partial functions $(f_n[t+n\varepsilon])$ admits a least upper bound (with respect to domain ordering) $\bar{f}$, which is a total function satisfying $\bar{f}R\bar{f}$ and $\bar{f}[t]=f[t]$.

5.3. Theorem. *Let G be a causal function from $\mathbb{F}$ to $\mathbb{F}$, and R be a total and overcausal binary relation on $\mathbb{F}$, such that*

1) $\exists t_0\in\mathbb{T}$ *such that* $\forall f\in\mathbb{F}, f[t_0]=G(f)[t_0]$,

2) $\forall f\in\mathbb{F}$, $G(f)\,Rf\Rightarrow f=G(f)$

Then, for every $f\in\mathbb{F}$, $fRf\Leftrightarrow G(f)\,Rf$.

Proof. The $\Leftarrow$ implication is a straightforward consequence of assumption (2). Let f such that fRf. Then $f[t_0]=G(f)[t_0]$, from assumption (1). Assume $f[t]=G(f)[t]$, and show $f[t+\varepsilon]=G(f)[t+\varepsilon]$: Since R is overcausal, we have $f'[t+\varepsilon]\,|\,fRf'\}=\{f'[t+\varepsilon]\,|\,G(f)\,Rf'\}$. Now, $f[t+\varepsilon]$ belongs to this set, since Rf. From the property (a) above, the relation $G\cdot R$ is overcausal, and there exists $\bar{f}$ such that $G(\bar{f})\,R\bar{f}$ and $\bar{f}[t+\varepsilon]=f[t+\varepsilon]$. From the assumption (2), $\bar{f}=G(\bar{f})$, and so $f[t+\varepsilon]=G(f)[t+\varepsilon]$ from the causality of G. So we have shown that fRf implies $f=G(f)$, which implies $G(f)\,Rf$.

This theorem may be applied to the introductory example of this section: Let R_1 be the relation performed by the system S, relating its input event e to its output, and let R_2 be the relation performed by the environment E, relating this output to the event e sent to S. The events that S may receive are the solutions of $e\,\mathrm{Re}$, where R is the product relation $R_1\cdot R_2$. Assume that R is overcausal, that is, assume that the "machines" S and E have a strictly positive response time. Let G be the filtering function $\lambda e\cdot e/C$, which is causal. Then the property P2 is equivalent to

$$G(e)\,Re\Rightarrow e=G(e)$$

and so, implies that the hypothesis of the theorem holds. So the theorem provides

$$e\,\mathrm{Re}\Leftrightarrow e=G(e)$$

which is the wanted property P1.

6.4. *Causality Analysis*

As it will appear in Part 2, our approach for specifying and constructing systems mainly consists of stating and handling relations over objects (events and variables), without paying much care of which of these objects are inputs or outputs. These relations have to be oriented only when the overcausality theorem must be invoked for proof purpose, or when an actual implementation is foreseen. Such an orientation can be seen as an abstraction (a mapping) of these relations to the causal $\rightarrow$ and overcausal $\twoheadrightarrow$ relations over objects:

A relation $e\,\mathrm{Re}'$ will induce the abstraction $e\rightarrow e'$ if and only if R is causal, and the abstraction $\twoheadrightarrow$ iff R is overcausal.

Properties. It comes, from the Definitions 6.1 and the Property 6.2a that:

- $\rightarrow$ is a preordering relation (reflexive and transitive);
- $\twoheadrightarrow$ is transitive;
- $\twoheadrightarrow$ is stronger than $\rightarrow$, that is $e\twoheadrightarrow e'\Rightarrow e\rightarrow e'$;
- $\twoheadrightarrow$ dominates $\rightarrow$, that is,

$$e\rightarrow e'\twoheadrightarrow e''\Rightarrow e\twoheadrightarrow e'',$$
$$e\twoheadrightarrow e'\rightarrow e''\Rightarrow e\twoheadrightarrow e''.$$

Examples. We do not intend here to provide a complete causality analysis, that would apply to all the relation constructors precedingly defined. We shall only give some straightforward examples, that will be useful in the sequel: Let e,f,g be events, and C a condition. Then,

- $e\leqq f$ induces both $e\rightarrow f$ and $f\rightarrow e$, since it relates the counters of e and f at the same time;
- $g=e+f$ induces $e\rightarrow g$ and $f\rightarrow g$;
- $f=e/C$ induces $e\rightarrow f$ and $C\rightarrow f$;
- $f=D^{\delta}e$, $\delta>0$, induces $e\twoheadrightarrow f$, since it relates the counter of f at time t with the counter of e at time $t-\delta$.

Part II. Application to the Specification and the Proof of a Distributed bus Arbiter

In order to illustrate the use of the formalism we have proposed in the first part of this paper, we shall apply it to the design of a hardware device, which is a distributed, asynchronous bus arbiter. This example is of manifold interest:

- The intuitive specifications of the problem are behavioural: their expression in terms of states and transitions would be rather awkward and unnatural. We shall try to show that the translation of the initial intention into formal specification is straightforward.
- However, we shall see that, when literally interpreted, this initial intention leads to unfeasible specifications. So this example illustrates the fact that, when writing formal specifications, one may be led to rectify or make precise a requirements document that may be uncomplete or ambiguous, and that these corrections may take place before the beginning of the actual design process.
- We shall describe two algorithmic solutions of the arbiter, whose correctness will be proved, thus widely experimenting our deduction tools.
- Wiring diagrams corresponding to these solutions will be given. From formal descriptions of the behaviour of hardware components (wires, gates, flip-flops, ...), the conformity of these diagrams to the algorithmic descriptions can be proved, but these proofs are not given here, for conciseness sake.

7. Specifications of the Arbiter

A bus connects n units $U_1, \ldots, U_n$, which are wanted to be provided with an arbitration device. The arbiter must warrant the exclusive access to the bus, together with a priority rule. Moreover, the arbiter must be distributed, in the sense that each unit has an arbitration element, and that the bus allocation results from the cooperation of these n elements, which don't make use of any centralized resource (such as shared clocks or memory, monitors, ...).

More precisely, when wanting to access the bus, an unit sends a request to its associated arbiter, and waits for an acknowledgement before accessing the bus. Each arbiter knows the requests of its proper unit, and the idleness state of the bus. Arbiters may communicate with each others, in order to emit acknowledgement signals warranting the following properties:

- Exclusivity: At each time, at most one unit is allowed to access the bus;
- Reactivity: An unit may be allowed to access the bus only if it asked for it;
- Priority respect: In the case of concurrent requests, the access must be granted to the asking unit of smallest index;
- Promptness: The bus may not remain both idle and requested.

Let us formalize these requirements in terms of events. In a first step, we shall describe the required behaviour without regards to feasibility constraints.

Taking into account these constraints, and particularly those resulting of the distributed feature of the arbitration, will lead us, afterwards, to weaken this first specification.

7.1. *First Specification*

Let us introduce the following events:

r_i $(i=1 \ldots n)$: reception of a request from U_i,

a_i $(i=1 \ldots n)$: emission of an acknowledgement to U_i,

f_i $(i=1 \ldots n)$: releasing of the bus by U_i.

The only assumption concerning the environment of the arbiter, is the "discipline" of the units, that is the fact that an unit only accesses the bus when allowed, and only releases the bus after having accessed it. So, if acc_i denotes the event "U_i accesses the bus", this hypothesis may be written:

$$\forall i=1 \ldots n,\ a_i \geqq \mathrm{acc}_i \geqq f_i.$$

In order to simplify, however, we shall not introduce the events acc_i, and write down the discipline assumption as follows:

$$[\mathrm{DIS}] \quad \forall i=1 \ldots n,\ a_i \geqq f_i.$$

Under this assumption, the unit U_i will be regarded as owning the bus at the instant t, if and only if its acknowledgements number $\mathit{count}(a_i)(t)$ is strictly greater than its releases number $\mathit{count}(f_i)(t)$. So we can write the arbiter requirements as follows:

- *Exclusivity.* The number of units owning the bus at the instant t is

$$\Sigma\, \mathit{count}(a_i)(t) - \Sigma\, \mathit{count}(f_i)(t)$$

(unless explicitly written, generalized summations are from 1 to n). So we can write the exclusivity constraint by mean of events inequality:

$$[\mathrm{EX}] \quad a_i \leqq \Sigma f_i + 1.$$

- *Reactivity*

$$[\mathrm{RE}] \quad \forall i=1 \ldots n,\ a_i \leqq r_i.$$

- *Priority.* Let us write that, when unit U_i is granted the bus, no unit with index less than i is requesting for the bus. Now, unit U_j is requesting for the bus at t if and only if its request number $\mathit{count}(r_j)(t)$ excedes its acknowledgement number $\mathit{count}(a_j)(t)$. So, we can write

$$[\mathrm{PY}] \quad 1 \leqq j < i \leqq n \Rightarrow \mathit{count}(r_j) \circ \mathit{last}(a_i) \leqq \mathit{count}(a_j) \circ \mathit{last}(a_i).$$

- *Promptness.* The bus is idle whenever

$$\Sigma\, \mathit{count}(f_i)(t) \geqq \Sigma\, \mathit{count}(a_i)(t).$$

It is requested whenever

$$\Sigma\, count\,(r_i)(t) > \Sigma\, count\,(a_i)(t).$$

From the expression of exclusivity and reactivity, it follows that:

- the bus busy at t iff

$$\Sigma\, count\,(a_i)(t) = \Sigma\, count\,(f_i)(t) + 1$$

- the bus is not requested at t iff

$$\Sigma\, count\,(a_i)(t) = \Sigma\, count\,(r_i)(t).$$

The promptness constraint requires that, at each instant, at least one of the above conditions must hold. So:

$$\forall t, \min(\Sigma\, count\,(f_i)(t) - \Sigma\, count\,(a_i)(t) + 1,$$
$$\Sigma\, count\,(r_i)(t) - \Sigma\, count\,(a_i)(t)) = 0$$

or, more succintely,

$$[\mathrm{PP}] \quad \Sigma a_i = inf(\Sigma f_i + 1, \Sigma r_i).$$

The four constraints EX, RE, PY and PP thoroughly determine the outputs (a_i) computed by the arbitration system, from its inputs (f_i) and (r_i). However, it appears that these specifications are not strictly feasible. As a matter of fact:

- the promptness constraint implies that the reaction of the arbiter to unforeseeable events (f_i) and (r_i) is instantaneous;

- as stated by PY, the priority rule implies that each arbiter have an exact knowledge of the requests of units of higher priority, which is unfeasible in a distributed system.

So, we must weaken the constraints PY and PP.

7.2. *Feasible Specifications*

We first change the promptness requirement into a constraint of response time. Let d be the event "request for arbitration", which happens whenever either a request happens when the bus is idle, or the bus is released when requested:

$$d = inf(\Sigma f_i + 1, \Sigma r_i).$$

The promptness constraint $(\Sigma a_i = d)$ is replaced by the following:

$$[\mathrm{RT}] \quad \Sigma a_i \geqq D^{\delta} d$$

which implies, under exclusivity and reactivity assumptions, that every arbitration request be followed, in a delay smaller than δ, by the sending of an acknowledgement. The response time δ is a parameter of the arbiter.

There are several ways for weakening the priority rule. The following one is (arbitrarily) choosen: An arbitration request, happening at the instant t, may result in access grantation to the unit U_i only if no unit with higher priority than U_i is requesting the bus at t. We write:

$$[\mathrm{PR}] \quad 1 \leqq j < i \leqq n \Rightarrow$$
$$count\,(a_j) \circ last\,(d) \circ last\,(a_i) = count\,(r_j) \circ last\,(d) \circ last\,(a_i)$$

which expresses that if U_j has a higher priority than U_i, then U_j never asks for the bus at the last arbitration request preceding an acknowledgement to U_i.

The specification of the arbiter may then be summarized as follows: "Under the discipline assumption DIS, build the events (a_i) satisfying the constraints EX, RE, RT and PR". However, the solutions we shall present now, will need some strengthening of the discipline assumption.

8. Travelling Token Algorithm

8.1. *Description of the Algorithm*

A well-known solution consists, whenever an arbitration is requested, of emitting a "token" which travels through the sequence of arbiters, from high to low priorities, until being picked up by an arbiter which unit asks for the bus. This unit is granted access.

Now, let j_i $(i = 1 \ldots n)$ be the event "arbiter i receives the token". When j_i occurs,

- if U_i asks for the bus, then a_i is issued;

- else, the token is sent to the next arbiter; let d_i be the corresponding event.

If ε is an upper bound of the transmission delay between two successive arbiters, the algorithm may be described as follows:

(1) $\forall i = 1 \ldots n,\ a_i = j_i / [count\,(r_i) \geqq count\,(a_i + 1)]$,

(2) $d_i = j_i / [count\,(r_i) \leqq count\,(a_i)]$,

(3) $d_{i-1} \geqq j_i \geqq D^{\varepsilon} d_{i-1}$,

(4) $d_0 = d = \inf(\Sigma r_i, \Sigma f_i + 1)$.

In fact, some additional assumptions must be done. The first one is necessary for satisfying the response time:

(5) $n\varepsilon \leqq \delta$.

Moreover, the proof reveals that it is necessary to assume that every transmission between arbiters, and every occupation of the bus by a unit take a strictly positive time. So it is assumed that:

(3') $\forall i = 1 \ldots n,\ j_i < d_{i-1}$

(DIS') $f_i < a_i$.

8.2. *Proof of Exclusivity and Reactivity*

Let us illustrate the style of reasoning used, by proving the exclusivity and the reactivity. The remainder of the proof of the travelling token algorithm is given in Appendix 1.

From (1) and (2), we get $a_i+d_i=j_i$ $(i=1\ldots n)$ and so, from (3), $j_i \geqq a_i+j_{i+1}$ $(i=1\ldots n-1)$. Summing these inequalities, for $i=1$ to $n-1$ provides

$$j_1 \geqq \Sigma a_i + d_n \geqq \Sigma a_i.$$

Let $a=\Sigma a_i$. From (3') and (DIS), we get

$$a \leqq j_1 < d_0 \leqq \Sigma f_i + 1 \leqq a+1$$

which implies exclusivity $(a \leqq \Sigma f_i + 1)$.

Moreover, it implies also $a<a+1$, that is the single occurrence of a, and therefore all the events a_i, j_i, d_i are single occurrent.

Now, (1) implies that, for every i in $\{1\ldots n\}$,

$$count(r_i) \circ last(a_i) \geqq count(a_i+1) \circ last(a_i)$$

and the single occurrence of a_i implies

$$count(a_i+1) \circ last(a_i) \geqq count(a_i).$$

So,

$$count(r_i) \circ last(a_i) \geqq count(a_i).$$

From the retractivity of $last(a_i)$,

$$count(r_i) \geqq count(r_i) \circ last(a_i)$$

and so, we get the reactivity, $r_i \geqq a_i$.

8.3. *Wiring Diagram*

The basic element of the hardware realization of the arbiter, is a device performing - within a response time - the filtering of an event by a condition (see Fig. 4, the meaning of the symbols is given by Fig. 3). An edge triggered flip-flop samples the value of the condition C on the rising edges E of the input E. After a delay ε, long enough for the flip-flop stabilization, and if two successive edges of E are always separated by a delay greater than ε, then the rising edges of F and G are respectively, the result of the filtering of E by C and by not C.

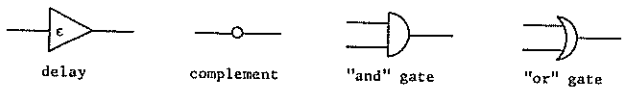

Fig. 3

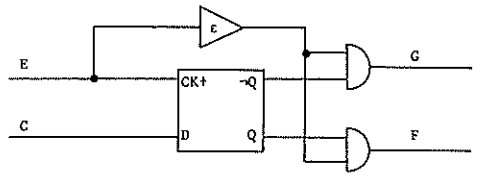

Fig. 4

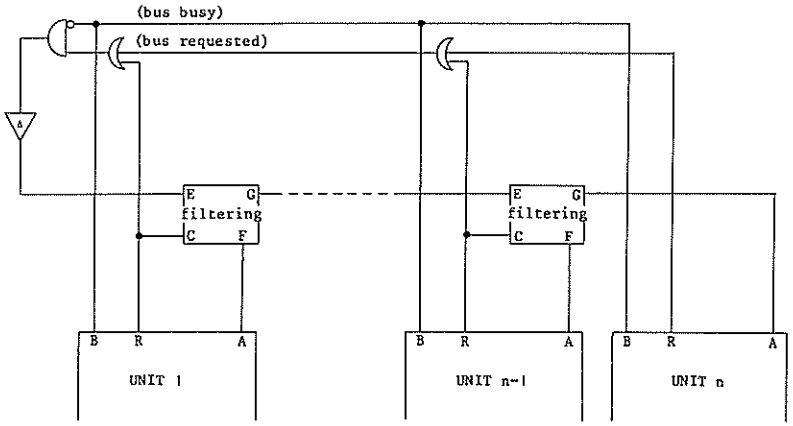

Fig. 5

Arbitration elements are built with such filtering devices. The entire diagram is shown in Fig. 5. The behaviour of units is assumed to be as follows:

- Sending a request consists of rising the output R;
- On receipt of an acknowledgement (input A high), the output R is reset, and the output B is rised;
- Releasing the bus consists of resetting B.

Under these assumptions, and from a description of the behaviour of hardware components, one could prove the conformity of this implementation to the algorithm.

9. Stabilized Condition Algorithm

As presented above, the travelling token algorithm consists of transmitting through the chain of arbiters, the event "request for arbitration", step by step filtered. Another, less standard, solution consists of transmitting through the chain of arbiters, only the condition involved in the filtering: The request for

arbitration is now straightly broadcasted to all arbiters. Each arbiter, when receiving such a request, displays its own opinion. After an adequate delay, the i-th arbiter may be sure that each arbiter with higher priority has received the request and has taken its decision, and so, the i-th arbiter may take its own decision.

This solution seems very close to the travelling token algorithm. In fact, it will be shown to be very different, both concerning its implementation and concerning its proof: while, in the travelling token, the most difficult property to prove was the priority, now exclusivity is the only difficult point. Since the proof will use the overcausality theorem, we shall indicate in the algorithm description, the causality relations linking together the objects involved in each constructing relations.

9.1. *The Algorithm*

• Let $d_0 = inf(\Sigma r_i, \Sigma f_i + 1)$ be the event "request for arbitration, and let d_i ($i = 1 \ldots n$) be the event "perception of d_0 by the i-th arbiter". This perception is assumed to occur strictly after each occurrence of d_0, but within a maximum delay $\varepsilon 1$:

$$d_0 > d_i \geqq D^{\varepsilon 1} d_0, \quad i = 1 \ldots n \qquad (f_i \rightarrow d_0 \text{ and } d_0 \rightarrow d_i).$$

• Let C_i be the requesting state of U_i, sampled on the occurrences of d_i:

$$C_i = [count\ell(r_i) \circ last\ell(d_i) \geqq count\ell(a_i + 1) \circ last\ell(d_i)]$$
$$(a_i \rightarrow C_i \text{ and } d_i \rightarrow C_i).$$

• After a delay $\varepsilon 2 > \varepsilon 1$, counted from an occurrence of d_i, the i-th arbiter is sure that all other arbiters have perceived the request for arbitration (i.e. d_j has happened, for every j), and so, have sampled and displayed their requesting states C_j. So, the i-th arbiter may take its granting decision:

$$\forall i = 1 \ldots n,\ a_i = b_i / A_i$$

where $b_i = D^{\varepsilon 2} d_i$ and $A_i = C_i \wedge \bigwedge_{j<i} \neg C_j$,

$$(d_i \twoheadrightarrow b_i,\ C_k \rightarrow A_i,\ b_i \rightarrow a_i,\ A_i \rightarrow a_i).$$

• The proof of this algorithm shows that it is correct only under the assumption that an unit which owns the bus, keeps it for a minimum delay $\varepsilon 3$, with $\varepsilon 3 > \varepsilon 1$:

$$\forall i = 1 \ldots n,\ f_i \leqq D^{\varepsilon 3} a_i,\ \varepsilon 3 > \varepsilon 1 \qquad (a_i \twoheadrightarrow f_i).$$

• The response time of the arbitration system is then $\varepsilon 1 + \varepsilon 2$, and, in order to satisfy the requirement RT, one must assume $\varepsilon 1 + \varepsilon 2 \leqq \delta$.

Exclusivity is proved in Appendix 2. It is the only difficult proof, since once t is assumed, other requirements are easy to deduce.

However, this proof is of different kind than the proof of the travelling oken algorithm: In the travelling token solution, the system clearly comes

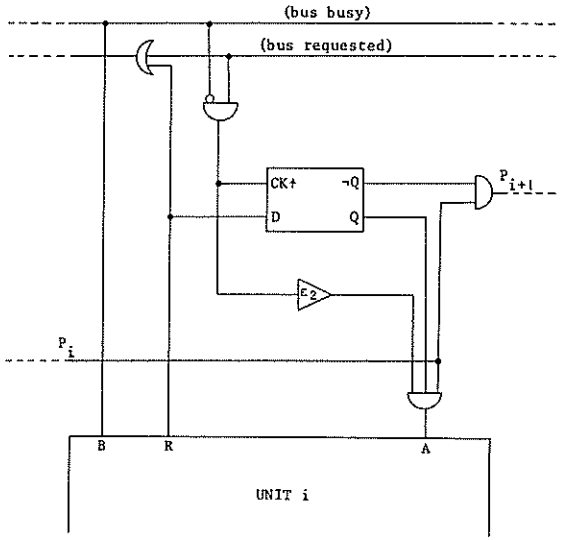

Fig. 6

back to an idle state after each arbitration. We had just to prove the required properties in each time interval between two successive idle states. But now, the behaviour of the arbiter may depend of its whole past history, and so its proof clearly needs induction. So, our overcausality theorem is used, in the following way:

• First, we show that, if all arbiters synchronously took their decisions (i.e. if all events b_i were simultaneous), the exclusivity would hold.

• Then, we consider an arbitration system, taking the lower bound of synchronous and asynchronous decisions. We show that exclusivity again holds in that case.

• At end, we show that the lower bound is always the asynchronous decision, and so, by the over-causality theorem, that the computation of this lower bound, and therefore the knowledge of the (unfeasible) synchronous decisions, are useless.

9.2. *Wiring Diagram*

Figure 6 shows a wiring diagram of the arbitration system using stabilized condition algorithm. The i-th arbiter receives the condition

$$P_i = \bigwedge_{j<i} \neg C_j \circ last\ell(d_j)$$

computed by its predecessors (P_1 is set to 1). The arbiter conjuncts this condition with its own term C_i – sampled by a flip-flop on each receipt of an arbitration request – and sends the result to the entry P_{i+1} of its successor. On the other hand, each arbitration request is delayed by $\varepsilon 2$, before instigating, when the input P_i is set and a request has been sampled, the sending of an acknowledgement.

This diagram shows that the response time of this solution is shorter than that of the travelling token, since an arbitration request crosses only one flip-flop.

Conclusion

We have illustrated our formalism on the design and proof of a distributed system. We hope this example showed that behavioural approach allows a quite natural specification (one only describes *what* the system must do, without describing *how* it will do). Another feature of our formalism is to be applicable from the initial specifications up to a level very close to the implementation, even in the case of hardware implementation.

Concerning validation tools, the proof of some apparently obvious properties may seem surprisingly difficult. Clearly, this difficulty is due to the very detailed level of the proof, but one of the conclusions we got from this exercise, is that this detailed level is necessary, since some other apparently obvious assertions about the arbiter, at last have been shown to be false during the proof.

Finally, an extended comparison with alternative approaches – such as temporal logics (for instance [10, 13]), calculi of synchronous processes [3, 12] or the event formalism of [7] – is still to be done, while briefly sketched in [8]. In our opinion, this comparison should bear on three main criteria:

the aptitude to accurately model actual systems: for instance, when dealing with asynchronous hardware systems, the approaches that take into account only a discrete notion of time, are likely to give rise to unnecessary modelling problems.

the aptitude to write natural and simple specifications: It is of well known experience that many design errors in computing systems, arise from incorrect specifications.

the aptitude to complete axiomatization and automatic proofs.

However, such a comparison is difficult to do, because these criteria, and mainly the first two, are essentially unquantifiable. In our model, these first two criteria have been deliberately privilegiated, at the expense of systematic proof capabilities, and at the opposite of most other works. The reasons of this choice are twofold:

On one hand, the undecidability of most proof problems in our field, is well-known.

- On the other hand, proving that a system description matches its specification is only interesting, in our opinion, if one can be reasonably convinced both that the description accurately captures the actual system behaviour, and that the specification properly expresses the informal intent of the system customer.

Appendix 1. Proof of the Travelling Token Algorithm

Priority and response time remain to be proved.

Priority

The proof of (PR) will be divided into several lemmas.

Lemma 1. *Unit U_k never asks for the bus at the last d_0 preceding d_k ($k=1\ldots n$): $\forall k=1\ldots n$,*

$$count(a_k)\circ last(d_0)\circ last(d_k)=count(r_k)\circ last(d_0)\circ last(d_k).$$

Lemma 2. *If $k<i$, there is an occurrence of d_k between j_i and the last d_0 preceding it: $1\leqq k<i\leqq n\Rightarrow$*

$$last(d_0)\circ last(j_i)=last(d_0)\circ last(d_k)\circ last(j_i).$$

Lemma 3. *No acknowledgement is issued between j_i and the last d_0 preceding it: $\forall i=1\ldots n$, $\forall k=1\ldots n$,*

$$count(a_k)\circ last(d_0)\circ last(j_i)=count(a_k)\circ last(j_i).$$

- First, we show that Lemmas 1 and 2 imply the priority constraint: From Lemma 2, and since $a_i \subseteq j_i$, if $1\leqq k<i\leqq n$ then:

$$\begin{aligned}
&count(a_k)\circ last(d_0)\circ last(a_i)\\
&\quad=count(a_k)\circ last(d_0)\circ last(d_k)\circ last(a_i)\\
&\quad=count(r_k)\circ last(d_0)\circ last(d_k)\circ last(a_i) \quad \text{(from Lemma 1)}\\
&\quad=count(r_k)\circ last(d_0)\circ last(a_i)\\
&\quad\text{(again from Lemma 2 and } a_i\subseteq j_i).
\end{aligned}$$

- Then we show that Lemma 3 implies Lemma 1: From the reactivity, we have, for every k in $\{1\ldots n\}$,

$$\begin{aligned}
&count(r_k)\circ last(d_0)\circ last(d_k)\\
&\quad\geqq count(a_k)\circ last(d_0)\circ last(d_k)\\
&\quad=count(a_k)\circ last(d_k) \quad \text{(from Lemma 3 and } d_k\subseteq j_k)\\
&\quad=count(d_k)\circ last(d_k) \quad \text{(from (2))}\\
&\quad\geqq count(r_k)\circ last(d_0)\circ last(d_k).\\
&\quad\text{(from the retractivity of } last(d_0)).
\end{aligned}$$

- Now, we prove Lemma 3: We have already shown $a<d_0<a+1$. So,

$$\begin{aligned}
count(a)\circ last(d_0)&\leqq count(d_0)\circ last(a)\circ last(d_0)\\
&\leqq count(d_0)\circ last(d_0)=(I-1)\circ count(d_0)
\end{aligned}$$

and

$$(I-1)\circ count(d_0)\leqq(I-1)\circ count(a+1)\circ last(d_0)$$
$$=count(a)\circ last(d_0).$$

So $count(a)\circ last(d_0)=(I-1)\circ count(d_0)$. Let us call this result Lemma 4.

Now, let i be in $\{1\ldots n\}$. On one hand, for every k in $\{1\ldots n\}$, since $j_k\leqq d_{k-1}$, we have:

$$count(j_k)\circ last(j_i)\leqq count(d_{k-1})\circ last(j_i)$$

and, on the other hand, from $j_i<d_{i-1}$, it follows that

$$count(j_i)\circ last(j_i)\leqq(I-1)\circ count(j_i)$$
$$\leqq(I-1)\circ count(d_{i-1})\circ last(j_i).$$

Summing these inequalities for $k=1$ to n provides

$$count(j_k)\circ last(j_i)\leqq(I-1)\circ\Sigma\, count(d_{i-1})\circ last(j_i).$$

Since $count(j_k)=count(a_k)+count(d_k)$ and since $count(d_n)\geqq 0$, the above inequality gives:

$$count(a)\circ last(j_i)\leqq(I-1)\circ count(d_0)\circ last(j_i)$$
$$=count(a)\circ last(d_0)\circ last(j_i)\quad\text{(from Lemma 4)}$$
$$\leqq count(a)\circ last(j_i)$$
(from the retractivity of $last(j_i)$)

and so, for every i,

$$count(a)\circ last(d_0)\circ last(j_i)=count(a)\circ last(j_i)$$

which implies Lemma 3.

• Lemma 2 remains to be proved. Let $1<k<i\leqq n$. We have only to show

$$count(d_0)\circ last(j_i)\leqq count(d_0)\circ last(d_k)\circ last(j_i)$$

Lemma 2 will follow by left composition by $time(d_0)$.

Since $j_i<d_{i-1}$ and $j_{i-1}=d_{i-1}+a_{i-1}$, we have

$$count(j_i)\leqq count(j_{i-1})\circ last(j_i)-count(a_{i-1})\circ last(j_i)$$

and since, for $k<i$,

$$count(j_{i-1})\leqq count(j_{i-2})-count(a_{i-2})\leqq\ldots\leqq count(d_k)-\sum_{m=k+1}^{i-2}count(a_m)$$

we get:

$$count(j_i)+\sum_{m=k+1}^{i-1}count(a_m)\circ last(j_i)\leqq count(d_k)\circ last(j_i).$$

In the same way,

$$count(d_k)+\sum_{m=1}^{k}count(a_m)\circ last(d_k)\leqq count(d_0)\circ last(d_k).$$

Finally, from

$$d_0\leqq j_i+\sum_{m=1}^{i-1}a_m+1$$

it follows that

$$count(d_0)\circ last(j_i)\leqq count(j_i)+\sum_{m=1}^{i-1}count(a_m)\circ last(j_i)$$
$$\leqq count(d_0)\circ last(d_k)\circ last(j_i).\quad\square$$

"*Response Time*"

Summing the relations $j_i=a_i+d_i\geqq D^{\varepsilon}d_{i-1}$ gives

$$a+d_n\geqq D^{n\varepsilon}d_0.$$

Since $n\varepsilon\leqq\delta$, we have only to show that $d_n=0$, that is to prove that whenever U_n receives a token, is asks for the bus:

$$count(r_n)\circ last(j_n)\geqq count(a_n+1)\circ last(j_n).$$

This relation may be easily derived, using the preceding lemmas. Indeed, by definition of d_0, when d_0 happens, at least one unit asks for the bus:

$$\Sigma\, count(r_i)\circ last(d_0)\geqq count(a+1)\circ last(d_0).$$

So

$$(I-1)\circ\Sigma\, count(r_i)\circ last(d_0)\circ last(j_n)\geqq count(a)\circ last(d_0)\circ last(j_n).$$

From the proof of priority,

$$\sum_{i<n}count(r_i)\circ last(d_0)\circ last(j_n)=\sum_{i<n}count(a_i)\circ last(d_0)\circ last(j_n).$$

So

$$(I-1)\circ count(r_n)\circ last(d_0)\circ last(j_n)\geqq count(a_n)\circ last(d_0)\circ last(j_n).$$

Now, from Lemma 3, the right hand side of the above inequality equals

$$count(a_n)\circ last(j_n)$$

and its left hand side, from the retractivity of $last(d_0)$ is less than

$$(I-1)\circ count(r_n)\circ last(j_n)$$

which terminates the proof.

Appendix 2. Proof of the Exclusivity of the Stabilized Condition Algorithm

a) Let $b_0=D^{\varepsilon 2}d_0$ and $a'_i=b_0/A_i$ (synchronous decisions). Since conditions A_i are exclusive, we get:

$$\Sigma a'_i=\Sigma(b_0/A_i)=b_0\Big/\bigvee_{i=1}^{n}A_i.$$

So

$$\Sigma a'_i\leqq b_0\leqq d_0\leqq\Sigma f_i+1.$$

and decisions a'_i respect exclusivity.

b) Now, let $a''_i=inf(a_i,a'_i)$, where a_i are the asynchronous decisions:

$$a_i=b_i/A_i\quad\text{with}\quad b_i=D^{\varepsilon 2}d_i.$$

We have

$$\Sigma a_i'' \leqq \Sigma a_i' \leqq b_0 \leqq d_0 \leqq \Sigma f_i + 1$$

hence, decisions a_i'' respect exclusivity.

c) Let us consider the system R that delivers the events (a_i), (a_i') as an undeterministic function of (a_i''), obtained by replacing

$$f_i \leqq D^{\varepsilon 3} a_i \quad \text{by} \quad f_i \leqq D^{\varepsilon 3} a_i'' \qquad (a_i'' \rightarrow\!\!\!\!\rightarrow f_i)$$

and the causal function F, computing $a_i'' = inf(a_i, a_i')$.

It comes from the causality analysis that R is overcausal ($a_i'' \rightarrow\!\!\!\!\rightarrow a_k$, $a_i'' \rightarrow\!\!\!\!\rightarrow a_k'$). Let us assume that there is a time origin t_0, such

$$\ell count(r_i)(t_0) = 0$$

which in turn yields

$$\ell count(a_i'')(t_0) = \ell count(a_i)(t_0).$$

If we show that $a_i \leqq a_i'$ and thus $a_i'' = a_i$, for each i, it will come from the overcausality theorem that the actual system

$$(a_i) R (a_i, a_i')$$

and the unfeasible one

$$(a_i'') R (a_i, a_i') \quad \text{and} \quad (a_i'') = F(a_i, a_i')$$

are equivalent (their outputs (a_i) are the same).

So it remains to show $a_i \leqq a_i'$, for each i. We have, for each j in $\{1 \ldots n\}$,

$$\begin{aligned} d_j < d_0 &\leqq 1 + \Sigma f_i \leqq 1 + D^{\varepsilon 3} \Sigma a_i'' \leqq 1 + D^{\varepsilon 3} \Sigma a_i' \\ &\leqq 1 + D^{\varepsilon 3} b_0 = 1 + D^{\varepsilon 2 + \varepsilon 3} d_0 \leqq 1 + D^{\varepsilon 2 + \varepsilon 3 - \varepsilon 1} d_j \end{aligned}$$

from which, since $\varepsilon 2 > \varepsilon 1$ and $\varepsilon 3 > \varepsilon 1$, it follows that

- d_j is single occurrent
- $d_0 \leqq 1 + D^{\varepsilon 2 + \varepsilon 3} d_0$
- for each i, j in $\{1 \ldots n\}$,

$$b_i = D^{\varepsilon 2} d_i \leqq D^{\varepsilon 2} d_0 < D^{\varepsilon 1} d_0 \leqq d_j \leqq d_0 \leqq 1 + D^{\varepsilon 2 + \varepsilon 3} d_0 \leqq 1 + D^{\varepsilon 1 + \varepsilon 2} d_0 \leqq 1 + b_i$$

hence $b_i < d_j \leqq 1 + b_i$, for each i in $\{0 \ldots n\}$, j in $\{1 \ldots n\}$. So,

$$actual(b_i) \circ \ell count(b_i) \leqq count(d_j) \leqq count(b_i + 1)$$

ınd, by right composition with $time(b_i)$, using

$$\ell count(b_i) \circ time(b_i) \geqq I$$

ınd

$$count(b_i + 1) \circ time(b_i) \leqq I$$

ve get

$$actual(b_i \leqq count(d_j) \circ time(b_i) \leqq actual(d_j).$$

Now, from $b_i \geqq D^{\varepsilon 1 + \varepsilon 2} d_j$, we get

$$actual(b_i) \geqq actual(D^{\varepsilon 1 + \varepsilon 2} d_j) = actual(d_j).$$

So $count(d_j) \circ time(b_i) = actual(d_j)$, and

$$count(d_j) \circ time(b_i) = count(d_j) \circ time(b_0)$$

from which it follows that, for every i, k in $\{1 \ldots n\}$,

$$A_k \circ time(b_i) = A_k \circ time(b_0).$$

A result on the subsequences (§ 5.3 d) may then be applied to provide:

$$b_i \leqq b_0 \Rightarrow a_i = b_i / A_i \leqq b_0 / A_i = a_i'$$

which is the wanted inequality.

References

1. Amblard, P., Caspi, P., Halbwachs, N.: Describing and reasoning about circuits behaviour by means of time functions. 1985 Conf. on Computer Hardware Description Languages, Tokyo, 39–48, August 1985
2. Ashcroft, E.A., Wadge, W.W.: LUCID: A non procedural language with iteration. CACM **20**, 519–526 (July 1977)
3. Austry, D., Boudol, G.: Algèbre de processus et synchronisation. TCS **30**, 91–131 (April 1984)
4. Bergerand, J.L., Caspi, P., Halbwachs, N.: Specification and formal validation of distributed systems: The real time approach. IEE Conf. Control 85, Cambridge, July 1985
5. Caspi, P., Halbwachs, N.: An approach to real time systems modelling. Proc. Int. Conf. on Distributed Computing Systems, Miami, 710–715, October 1982
6. Caspi, P., Halbwachs, N.: Conception certifiée de systèmes temporisés: Un exemple. Final Report of ATP-PCS, CNRS, September 1983
7. Chen, B.T., Yeh, R.T.: Formal specification and verification of distributed systems. Proc. Int. Conf. on Distributed Computing System, Miami, 380–387, October 1982
8. Halbwachs, N.: Modélisation et analyse des systèmes informatiques temporisés. Doctoral Thesis, University of Grenoble, June 1984
9. Kahn, G.: The semantics of a simple language for parallel processing. Proc. IFIP Congress 1974
10. Koymans, R., DeRoever, W.P.: Examples of real time temporal logic specification. Workshop on the analysis of Concurrent Systems, Cambridge, September 1983
11. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. CACM **21**, 558–565 (July 1978)
12. Milner, R.: Calculi for synchrony and asynchrony. TCS **25**, 267–310 (July 1983)
13. Moszkowski, B.: A temporal logic for multi-level reasoning about hardware. Proc. 6th Int. Symp. on Computer Hardware Description Languages, Pittsburgh, 79–90, May 1983
14. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. TCS, **13**, 85–108 (1981)
15. Sanchis, L.E.: Data types as lattices: Retractions, closures and projection. RAIRO Theor. Comput. Sci. **11**, 329–344 (1977)

Received October 31, 1984 / August 9, 1985