Causal Time Calculus

Franck Pommereau

Proc. of FORMATS'03, LNCS, Springer, 2003.

Causal Time Calculus

Franck Pommereau

LACL, Université Paris 12
61, avenue du général de Gaulle
94010 Créteil — France
pommereau@univ-paris12.fr

Abstract. We present a process algebra suitable to the modelling of timed concurrent systems and to their efficient verification through model checking. The algebra is provided with two consistent semantics: a structural operational semantics (as usual for process algebras) and a denotational semantics in terms of Petri nets in which time is introduced through counters of explicit clock ticks. This way of modelling time has been called *causal time* so the process algebra is itself called the *Causal Time Calculus* (*CTC*). It was shown in a separate paper [3] that the causal time approach allowed for efficient verification but suffered from a sensitivity to the constants to which counts of ticks are compared. We show in this paper how this weakness can be removed.

1 Introduction

This paper presents a process algebra in which the representation of timing constraints can be explicitly included. With respect to the many such models already defined (a short comparison is given in the conclusion), our contribution is: first, to provide a concurrent semantics instead of the interleaving generally used; second, to propose a multiway communication scheme; and third, to give a way through which efficient model checking can be performed. We thus define a structural operational semantics (SOS), explicitly including concurrency, through SOS rules in Plotkin's style [17]; then a consistent denotational semantics is given by a transformation from process terms to Petri nets on which dedicated verification techniques may be applied [8]. The involved class of Petri nets consists in composable, labelled and coloured nets in which time is introduced by explicitly modelling clocks and counters of clock ticks. This is sometimes called the *causal time* approach and thus, our algebra is called the *Causal Time Calculus* (*CTC*). It is worth noting that CTC is actually a descendant of the *Petri Box Calculus* [2] and inherits, in particular, a large part of its syntax, the multiway communication scheme and the concurrent semantics.

A case study [3] made a comparison between the causal time approach and timed automata [1]; it turned out that the verification of Petri nets with causal time using a general model checker for high-level Petri nets (MARIA [13]) was more efficient than the verification of timed automata using well known tools (Kronos [20] and UPPAAL [12]). The approach in [3] was to translate timed

automata into the closest possible Petri nets, without any special optimisation. Indeed, an important concern was to avoid a biased comparison. Thus, even if one case study does not allow for any conclusion, this result is very encouraging. However, the causal time approach suffers from a sensitivity to the constants to which ticks counters are compared. The size of the state space actually depends on the product of the largest constants compared to each counter. If one uses k counters, each compared to a value n, one gets n^k states only to represent the timing information. We show at the end of this paper that this problem can be removed by identifying states which differ by the values of the ticks counters but are otherwise identical, *i.e.*, lead to the same evolutions. This is very similar to the notion of regions developed for timed automata [1] and allows to use verification techniques based on the concurrent semantics of Petri nets [8] which are generally much more efficient than those based on the interleaving semantics (as in MARIA). The benefits is thus twofold: first, to remove the sensitivity to constants, and second, to improve the good performances obtained in [3].

The next section defines the algebra of terms and its operational semantics. The section 3 presents the Petri nets, called *boxes*, used to define the denotational semantics and gives the transformation from process terms to boxes. These two sections form an extended abstract of the technical report [18] which provides the full definitions, properties and proofs. The section 4 addresses the question of the verification of boxes and is a completely new contribution. We conclude in the section 5 and briefly compare CTC to other timed process algebras.

2 CTC terms: syntax and operational semantics

Communication. We assume that there is a set $\mathbb{A}$ of *actions* used to model handshake communication. We also assume that $\tau \notin \mathbb{A}$ and that, for every $a \in \mathbb{A}$, $\widehat{a}$ is also an action in $\mathbb{A}$ such that $\widehat{\widehat{a}} = a$. A *multiaction* is a finite multiset of actions and we denote by $\{\}$ the empty multiaction.

Communication in CTC generalises the synchronisation of CCS [16] (allowed by the parallel composition) followed by the restriction (which forbids the independent execution of synchronised actions). This is formalised through the partial functions $\varphi_{\mathsf{sc}\,a}$, for $a \in \mathbb{A}$, which map the multisets of multiactions allowed to handshake to the multiactions resulting from the communication. For instance, $\varphi_{\mathsf{sc}\,a_1}$ is such that its domain contains $\Gamma \stackrel{\text{df}}{=} \{\{a_1, a_1, a_2\}, \{\widehat{a_1}, a_3\}, \{\widehat{a_1}\}\}$, which denotes that the multiactions of Γ may perform a three-way synchronisation. The multiaction corresponding to this communication is given by $\varphi_{\mathsf{sc}\,a_1}(\Gamma) \stackrel{\text{df}}{=} \{a_2, a_3\}$. On the contrary, $\{\{a_1, a_2\}, \{a_2\}\}$ is not in the domain of $\varphi_{\mathsf{sc}\,a_1}$ because the multiactions $\{a_1, a_2\}$ and $\{a_2\}$ cannot handshake.

Clocks. The progression of time will be reflected on *clocks* which are nonnegative integer variables that can be tested or updated by the processes. We denote by $\mathbb{N}$ the set of natural numbers. The set $\mathbb{C}$ of clocks is finite and we assume that there exists a function $\mathsf{max} : \mathbb{C} \to \mathbb{N} \setminus \{0\}$ which gives the maximum value allowed for each clock. This allows to specify *deadlines*, *i.e.*, time boundaries within

which a process completes [7]. Time progresses when a *tick* occurs incrementing simultaneously all the clocks, which is forbidden when at least one clock has reached its maximum. Notice that we require $\mathsf{max}(c) > 0$ for all $c \in \mathbb{C}$, otherwise no tick could ever occur, resulting in an untimed model.

A *clock vector* is a partial function $\theta : \mathbb{C} \to \mathbb{N}$ such that for all $c \in \mathsf{dom}(\theta)$, $\theta(c) \leq \mathsf{max}(c)$. Such a mapping associates its current value to each clock c in its domain, *i.e.*, the number of ticks which occurred since the last reset of c. We denote by $\mathbb{V}$ the set of all clock vectors. For θ_1 and θ_2 in $\mathbb{V}$ such that $\mathsf{dom}(\theta_1) \cap \mathsf{dom}(\theta_2) = \varnothing$, we denote by $\theta_1 + \theta_2$ the clock vector whose domain is $\mathsf{dom}(\theta_1) \cup \mathsf{dom}(\theta_2)$ and which is equal to θ_1 on $\mathsf{dom}(\theta_1)$ and to θ_2 on $\mathsf{dom}(\theta_2)$. By extension, writing $\theta_1 + \theta_2$ will implicitly imply that the domains of θ_1 and θ_2 are disjoint. In the following we denote by $\theta(e)$ the evaluation of the expression e in which the clocks have been replaced by their values as specified by θ. For instance, if $\theta(c) = 3$ then we have $\theta(c + 1) = 4$.

Clocks vectors will be handled through *clock expressions*, attached to the atomic process terms, which are sets of expressions of two kinds: *comparisons* (for instance $c_1 + c_2 > 3$), used to specify a condition under which an atomic process may be executed; and *assignments* (for instance $c_1 := c_2 + 1$) which allow to change the value of a clock. It is required that a clock expression δ contains at most one assignment for each clock in $\mathbb{C}$. A particular clock expression will be used in the following to represent the occurrence of a tick: $\delta_\tau \stackrel{\text{df}}{=} \{c := c + 1 \mid c \in \mathbb{C}\}$. We say that a clock vector θ *enables* a clock expression δ if (1) all the clocks involved in δ belong to the domain of θ, (2) all the comparisons in δ evaluate to true through θ, and (3) all the assignments $c := e$ in δ are such that $\theta(e) \leq max(c)$. In such a case, applying to θ all the assignments specified in δ leads to a new vector which is denoted $\delta(\theta)$.

Syntax. The syntax of CTC is given in the figure 1. We distinguish *static* terms, denoted by E, which cannot evolve, from *dynamic* ones, denoted by D, where the current state of the execution is represented by overbars (initial state) and underbars (final state) which may flow through the terms during their execution. We denote by F a static or dynamic term.

The atomic terms are of the form $\alpha\delta$ where α is a multiaction and δ is a clock expression. Consider for instance the two atomic terms $\{\widehat{a_1}, a_2\}\{\}$ and $\{a_1\}\{c_2 > 0, c_2 := 0\}$. The first one denotes the simultaneous receiving of the signal a_1 and sending of the signal a_2, which is untimed; the second one can send the signal a_1 and reset the clock c_2 if the value associated to c_2 is greater than zero. Various operators allow to combine terms:

$$
\begin{array}{rcl}
E & ::= & \alpha\delta \mid E \,\mathsf{sc}\, a \mid E \| E \mid E \fatsemi E \mid E \,\Box\, E \mid E \circledast E \mid E \,@\, \theta \\
D & ::= & \overline{E} \mid \underline{E} \mid D \,\mathsf{sc}\, a \mid D \| D \mid D \fatsemi E \mid E \fatsemi D \\
 & \mid & D \,\Box\, E \mid E \,\Box\, D \mid D \circledast E \mid E \circledast D \mid D \,@\, \theta
\end{array}
$$

Fig. 1. The syntax of CTC terms, where $\alpha\delta$ is an atomic term, $a \in \mathbb{A}$ and $\theta \in \mathbb{V}$.

– the *sequential composition* $F_1 \fatsemi F_2$ (F_1 is executed first and followed by F_2) may be seen as a generalisation of the prefixing operator used in CCS;
– the *choice* $F_1 \Box F_2$ (either F_1 or F_2 may be executed) corresponds to the choice of CCS;
– the *parallel composition* $F_1 \| F_2$ (F_1 and F_2 may evolve concurrently) differs from that used in CCS since it does not allow for synchronisation;
– the *iteration* $F_1 \circledast F_2$ (F_1 is executed an arbitrary number of times and is followed by F_2) allows to represent repetitions while in CCS the recursion would be used;
– the *scoping* $F \,\mathsf{sc}\, a$ (all the handshakes involving a and $\widehat{a}$ are enforced) was discussed above.

In order to model the clocks, terms are decorated with clock vectors. For instance, we may form $\{a\}\{c := 0\} @ \{c \mapsto 5\}$ denoting that the clock c has value the 5 for the atomic term $\{a\}\{c := 0\}$.

Operational semantics. An important part of the operational semantics relies on equivalence rules allowing to identify distinct terms which actually correspond to the same state. Formally, we define $\equiv$ as the least equivalence relation on terms such that all the rules in the figure 2 are satisfied. Consider for instance the rule IS2: it states that having the first component of a sequence in its final state is equivalent to have the second component in its initial state, which is indeed the expected semantics of a sequential composition.

Contrasting with CCS where evolutions are expressed by removing prefixes of terms, like in $a.P \xrightarrow{a} P$, the structure of CTC terms never evolves; instead, the overbars may be changed to underbars as in

$$\overline{\{a\}\{c := 0\}} @ \{c \mapsto 5\} \xrightarrow{(\{a\},\{c \mapsto 5\})} \underline{\{a\}\{c := 0\}} @ \{c \mapsto 0\}$$

which produces the *timed multiaction* $(\{a\}, \{c \mapsto 5\})$ denoting the occurrence of $\{a\}$ when the clock c had the value 5, while the reset $c := 0$ has been reflected on the new clock vector. In order to have a concurrent semantics, several timed multiactions may be combined, denoting their concurrent occurrence. Given Γ_1 and Γ_2 two multisets of multiactions and $\theta_1, \theta_2 \in \mathbb{V}$ having disjoint domains, $A_1 \stackrel{\text{df}}{=} (\Gamma_1, \theta_1)$ and $A_2 \stackrel{\text{df}}{=} (\Gamma_2, \theta_2)$ are *timed multisets of multiactions* and $A_1 + A_2 \stackrel{\text{df}}{=} (\Gamma_1 + \Gamma_2, \theta_1 + \theta_2)$.

Then, we define a ternary relation $\longrightarrow$ as the least relation comprising all (F, A, F') where F and F' are terms and A is a timed multiset of multiactions, such that the rules in the figure 3 hold. Notice that we use $F \xrightarrow{A} F'$ to denote $(F, A, F') \in \longrightarrow$. When used with $\diamond = \|$, the rule EOP is the way through which true concurrency is introduced. When used with another operator, the syntax ensures that at most one of A_1 or A_2 has a nonempty multiset of multiactions since at least one of the operands must be a static term. In these cases, the rule EOP shall be used in conjunction with EQ1 in order to compose a static term with a dynamic one. Concerning the rule ETICK, it should be noted that the side condition "θ enables δ_τ" implies that $\mathsf{dom}(\theta) = \mathbb{C}$; thus the occurrence of a tick always simultaneously increments all the clocks.

EX	$\dfrac{E \equiv E'}{\underline{E} \equiv \underline{E'}}$	ENT	$\dfrac{E \equiv E'}{\overline{E} \equiv \overline{E'}}$
CON1	$\dfrac{F \equiv F'}{F \,\mathsf{sc}\, a \equiv F' \,\mathsf{sc}\, a}$	CON2	$\dfrac{F_1 \equiv F_1', F_2 \equiv F_2'}{F_1 \diamond F_2 \equiv F_1' \diamond F_2'}$
ISC1	$\overline{E \,\mathsf{sc}\, a} \equiv \overline{E \,\mathsf{sc}\, a}$	ISC1	$\underline{E \,\mathsf{sc}\, a} \equiv \underline{E \,\mathsf{sc}\, a}$
IPAR1	$\overline{E_1 \| E_2} \equiv \overline{E_1} \| \overline{E_2}$	IPAR2	$\underline{E_1 \| E_2} \equiv \underline{E_1} \| \underline{E_2}$
IC1L	$\overline{E_1 \,\Box\, E_2} \equiv \overline{E_1} \,\Box\, E_2$	IC2L	$\underline{E_1 \,\Box\, E_2} \equiv \underline{E_1} \,\Box\, E_2$
IC1R	$\overline{E_1 \,\Box\, E_2} \equiv E_1 \,\Box\, \overline{E_2}$	IC2R	$\underline{E_1 \,\Box\, E_2} \equiv E_1 \,\Box\, \underline{E_2}$
IS1	$\overline{E_1 \fatsemi E_2} \equiv \overline{E_1} \fatsemi E_2$	IS2	$\underline{E_1} \fatsemi E_2 \equiv E_1 \fatsemi \overline{E_2}$
IS3	$E_1 \fatsemi \underline{E_2} \equiv \underline{E_1 \fatsemi E_2}$	IIT1	$\overline{E_1 \circledast E_2} \equiv \overline{E_1} \circledast E_2$
IIT2	$\underline{E_1} \circledast E_2 \equiv \overline{E_1} \circledast E_2$	IIT3	$\underline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$
IIT4	$\overline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$	IIT5	$E_1 \circledast \underline{E_1} \equiv \underline{E_1 \circledast E_2}$
IAT1	$\dfrac{F \equiv F'}{F \mathbin{@} \theta \equiv F' \mathbin{@} \theta}$	IAT2	$\overline{E \mathbin{@} \theta} \equiv \overline{E} \mathbin{@} \theta$
IAT3	$\underline{E \mathbin{@} \theta} \equiv \underline{E} \mathbin{@} \theta$	IAT4	$(F \,\mathsf{sc}\, a) \mathbin{@} \theta \equiv (F \mathbin{@} \theta) \,\mathsf{sc}\, a$
IAT5	$(F_1 \diamond F_2) \mathbin{@} (\theta_1 + \theta_2) \equiv (F_1 \mathbin{@} \theta_1) \diamond (F_2 \mathbin{@} \theta_2)$		

Fig. 2. Similarity relation, where $a \in \mathbb{A}$, $\diamond \in \{\|, \fatsemi, \Box, \circledast\}$ and $\{\theta, \theta_1, \theta_2\} \subset \mathbb{V}$.

EQ1	$F \xrightarrow{(\{\},\theta_\varnothing)} F$	where $\mathsf{dom}(\theta_\varnothing) = \varnothing$
EQ2	$\dfrac{F \equiv F' \;,\; F' \xrightarrow{A} F'' \;,\; F'' \equiv F'''}{F \xrightarrow{A} F'''}$	
EAT	$\dfrac{F \xrightarrow{(\Gamma,\theta_1)} F'}{F \mathbin{@} \theta_2 \xrightarrow{(\Gamma,\theta_1+\theta_2)} F' \mathbin{@} \theta_2}$	
EA	$\overline{\alpha\delta} \mathbin{@} \theta \xrightarrow{\{(\alpha,\theta)\}} \underline{\alpha\delta} \mathbin{@} \delta(\theta)$	if θ enables δ
ETICK	$\dfrac{F \mathbin{@} \theta \xrightarrow{A} F' \mathbin{@} \theta}{F \mathbin{@} \theta \xrightarrow{A+(\{\tau\},\theta)} F' \mathbin{@} \delta_\tau(\theta)}$	if θ enables δ_τ
ESC	$\dfrac{F \xrightarrow{(\Gamma_1,\theta_1)+\cdots+(\Gamma_k,\theta_k)} F'}{F \,\mathsf{sc}\, a \xrightarrow{(\varphi_{\mathsf{sc}\,a}(\{\Gamma_1,\ldots,\Gamma_k\}),\theta_1+\cdots+\theta_k)} F' \,\mathsf{sc}\, a}$	if τ does not appear in any Γ_i and $\{\Gamma_1, \ldots, \Gamma_k\} \in \mathsf{dom}(\varphi_{\mathsf{sc}\,a})$
EOP	$\dfrac{\underline{F_1} \xrightarrow{A_1} F_1' \;,\; \underline{F_2} \xrightarrow{A_2} F_2'}{F_1 \diamond F_2 \xrightarrow{A_1+A_2} F_1' \diamond F_2'}$	if τ does not appear in A_1 neither in A_2

Fig. 3. Evolution rules, where $\alpha\delta$ is an atomic term, $\{\theta, \theta_1, \theta_2, \theta_\varnothing\} \subset \mathbb{V}$, $a \in \mathbb{A}$, $\diamond \in \{\|, \fatsemi, \Box, \circledast\}$ and assuming that all the applications of $+$ are well defined.

3 Denotational semantics

The algebra of boxes. We start by introducing the labelled coloured Petri nets called *boxes* and the operations used to compose them. These operations exactly correspond to those defined on terms: for each operator on terms, there exists a similar operator defined on boxes.

The labelling of boxes allows to distinguish the *entry*, *internal* and *exit* places; all together, they are called the *control* places since their role is to represent the current state of the control flow. The marking of the entry places corresponds to the initial marking of a box and thus we define $\overline{N}$ as the box N in which one token is added to each entry place. Similarly, the exit places correspond to the final marking and $\underline{N}$ is defined as expected. The internal places correspond to intermediate states during the execution of a box. Except for the scoping, the operators of CTC are also based on the labels of places. For instance, the sequential composition $N_1 \fatsemi N_2$ is defined by combining the exit places of N_1 with the entry places of N_2, resulting in internal places whose marking represent both the final marking of N_1 and the initial one of N_2.

Another class of places is distinguished thanks to their labels, these are the *clock* places in which clock values are modelled. A box has exactly one clock place labelled by c for each $c \in \mathbb{C}$. When several nets are combined, for instance using the parallel composition, clock places with the same label are automatically merged (with their markings) ensuring a unique representation of each clock. While the control places are only allowed to carry the ordinary black token $\bullet$, each clock place may carry any integer from $\mathbb{N}$. Thus, the clock places are the only coloured ones.

The labelling of an arc consists in a multiset of values, variables or expressions which represents the tokens flowing on the arc when the attached transition is executed.

The labelling of a transition contains a multiaction as in atomic terms. This allows to define the scoping w.r.t. $a \in \mathbb{A}$ whose role is to merge sets of transitions whose labels $\alpha_1, \ldots, \alpha_k$ belong to the domain of $\varphi_{\mathsf{sc}\,a}$, the newly created transition being labelled by $\varphi_{\mathsf{sc}\,a}(\{\alpha_1, \ldots, \alpha_k\})$. Transitions are also labelled by guards (boolean conditions involving the variables used in adjacent arcs) which must evaluate to true in order to allow the execution of the transitions. When this occurs, the variables in the guard and the adjacent arcs are associated to values through a *binding* σ and we denote by t_σ the occurrence of t under the binding σ.

From terms to boxes. Let $\alpha\delta$ be an atomic term, its denotational semantics is given by the box $N_{\alpha\delta}$ defined as follows. Its places are: one entry place s_e, one exit place s_x, and one clock place s_c labelled by c for each $c \in \mathbb{C}$. The marking is empty for the control places and $\{0\}$ for the clock places. The box has one transition t labelled by $\{\tau\}(\bigwedge_{c\in\mathbb{C}} c < \mathsf{max}(c))$ which models the tick and, for each $c \in \mathbb{C}$, there is one arc labelled by $\{c\}$ from s_c to t and one arc labelled by $\{c+1\}$ from t to s_c. There is also one transition u labelled by $\alpha\gamma$, where γ is the disjunction of all the comparisons in δ (or true if there is no comparison

in δ), which models the atomic action $\alpha\delta$. This transition u has one incoming arc from s_e labelled by $\{\bullet\}$ and one outgoing arc to s_x with the same label. The other arcs on u correspond to the clocks involved in δ; for all $c \in \mathbb{C}$:

- if c appears in δ in comparisons only, then there is one arc from s_c to u labelled by $\{c\}$ and one arc from u to s_c with the same label;
- if c appears in δ in an assignment $c := e$, where e is an expression, then there is one arc from s_c to u labelled by $\{c\}$ and one arc from u to s_c with the label $\{e\}$;
- if c does not appear in δ then there is no arc between s_c and u.

The denotational semantics is then defined by induction:

$$\mathsf{box}(\alpha\delta) \stackrel{\text{df}}{=} N_{\alpha\delta} \qquad \mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \qquad \mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)}$$

$$\mathsf{box}(F\ \mathsf{sc}\ a) \stackrel{\text{df}}{=} \mathsf{box}(F)\ \mathsf{sc}\ a \qquad \mathsf{box}(F_1 \diamond F_2) \stackrel{\text{df}}{=} \mathsf{box}(F_1) \diamond \mathsf{box}(F_2)$$

where $\alpha\delta$ is an atomic term, $a \in \mathbb{A}$ and $\diamond \in \{\|, \fatsemi, \Box, \circledast\}$. Moreover, for $\theta \in \mathbb{V}$, $\mathsf{box}(F \mathbin{@} \theta)$ is $\mathsf{box}(F)$ in which the marking of each clock place labelled by $c \in \mathsf{dom}(\theta)$ is set to $\{\theta(c)\}$. For example, assuming $\mathbb{C} \stackrel{\text{df}}{=} \{c\}$ and $\mathsf{max}(c) \stackrel{\text{df}}{=} 4$, the box on the left of the figure 4 is

$$\mathsf{box}\left(\overline{\{a_1\}\{c := 0\} \fatsemi \{a_2\}\{c \geq 2\}}\right) \quad .$$

The operational and denotational semantics are closely related: they are actually consistent in arguably the strongest sense since a term and the corresponding box generate isomorphic transitions systems.

4 Verification through unfoldings

A well known technique to perform efficient model checking on Petri nets is to use their concurrent semantics expressed by prefixes of their *unfolding* which are also Petri nets, see [8]. The traditional definition of unfoldings is based on low-level Petri nets, but it was shown in [10] that coloured Petri nets like boxes may be unfolded as well (producing a low-level net). An example of a box and a prefix of its unfolding is given in the figure 4.

In the unfolding, places are called *conditions* and are labelled by the name and the marking of the place to which they correspond in the original net; similarly, transitions are called *events* and correspond to the transition occurrence which labels them. The labelling function is an *homomorphism* and will be denoted by h in the following. In the figure 4, this labelling is indicated inside the nodes and is simplified: a condition labelled by an integer n denotes the presence of n in the place s_c and conditions labelled by s_1, s_2 or s_3 denote the token $\bullet$ in the corresponding place.

An unfolding may be executed by putting one token in each condition with no predecessor. One can check on the example that this allows to perfectly mimic the behaviour of the original net. Notice that, when the pairs of conditions depicted

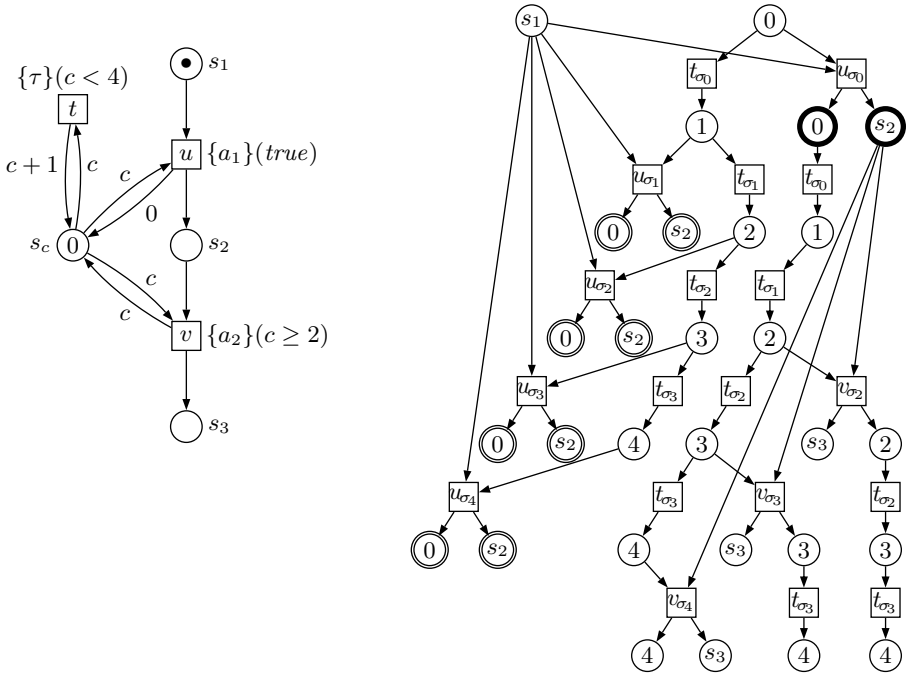

Fig. 4. The box of $\overline{\{a_1\}\{c := 0\} \fatsemi \{a_2\}\{c \geq 2\}}$ (on the left) and a prefix of its unfolding (on the right), where $\sigma_i \stackrel{\text{df}}{=} \{c \mapsto i\}$ for $0 \leq i \leq 4$; assuming $\mathbb{C} \stackrel{\text{df}}{=} \{c\}$ and $\mathsf{max}(c) \stackrel{\text{df}}{=} 4$.

with double lines become marked, the execution may be continued from the conditions depicted with thick lines. Indeed, these double-lined pairs are *cuts* where the unfolding have been truncated since the corresponding markings were already represented by the thick-lined pair of conditions. This allows to consider only *prefixes* of the full unfolding which may be itself infinite (if the net has an infinite run). Such a prefix is *complete* (w.r.t. reachability properties) if every reachable marking of the original net is represented in the prefix. This guarantees that reachability properties can be verified on the prefix rather than on the original net.

The notion of completeness actually depends on the properties that should be verified. Usually, those related to reachability are considered, but different ones may be envisaged like in [9]. In our case, if only control flow properties have to be verified on a box, the occurrences of ticks and the markings of clock places could be removed from the unfolding of this box. In the following, we present an intermediate simplification which keeps some timing information but without its full precision: it will not be possible to exactly know the values of the clocks when an event occurs; instead, we will obtain a range of possible values. Moreover, in the simplified unfolding, an event labelled by an occurrence of the tick transition will denote that "time is passing" instead of the more accurate "one tick occurs".

Simplification of unfoldings. We now show how to collapse chains of ticks (as, *e.g.*, at the bottom-right of the figure 4) thus removing the sensitivity of model checking to the constants used in clock expressions. It should be stressed that, for practical applications, the transformation described below should be applied on-the-fly during the computation of the unfolding; but, the principle being independent of the algorithm actually used, we prefer the current presentation.

Let x be an event or a condition, we denote by ${}^\bullet x$ the set of nodes immediately preceding x and by $x^\bullet$ those immediately succeeding x. This notation naturally extends to sets of nodes. For a set E of events, we denote by $\mathsf{trans}(E)$ the multiset of transitions involved in E, *i.e.*,

$$\mathsf{trans}(E) \stackrel{\text{df}}{=} \sum_{e \in E \wedge h(e) = w_\sigma} \{w\} \quad .$$

To start with, we change the labelling of conditions to triples (s, p, q) where s is a place of the original net and p, q are integers such that $0 \leq p \leq q$. If s is a control place, this label indicates that the condition corresponds to s marked by $\bullet$; but if s is a clock place, the condition corresponds to the place s whose marking is any integer in $\{p, \ldots, q\}$. So, the labelling is changed as follows: for each condition which corresponds to the marking of the control place s, the label becomes $(s, 0, 0)$; for each condition which corresponds to the marking of the clock place s' by the integer n, the label is changed to (s', n, n).

Then, we consider an event e labelled by an occurrence t_σ of the tick transition. We call e a *tick event*. One can show that, if ${}^\bullet e = \{c_1, \ldots, c_k\}$ with $h(c_i) = (s_i, p_i, q_i)$ for $1 \leq i \leq k$, then, because the tick transition is connected to clock places through side loops (and not connected to any other place), we must have $e^\bullet = \{c'_1, \ldots, c'_k\}$ and $h(c'_i) = (s_i, p'_i, q'_i)$ for $1 \leq i \leq k$. We distinguish two sets of events: $E \stackrel{\text{df}}{=} ({}^\bullet e)^\bullet$ which contains all the events in conflict with e (including e) and $E' \stackrel{\text{df}}{=} (e^\bullet)^\bullet$ which contains all the events enabled by the occurrence of e. Then, if $\mathsf{trans}(E) = \mathsf{trans}(E')$, it means that the tick do not change the enabling in the net (it may change the bindings but not the transitions which are enabled). So, e is removed and the conditions in $e^\bullet$ are merged to those in ${}^\bullet e$. Each condition c'_i (whose label is (s_i, p'_i, q'_i)) is merged to the corresponding c_i (labelled by (s_i, p_i, q_i)) as follows:

- the condition c'_i is removed and the label of c_i is changed to (s_i, p_i, q'_i);
- each tick event $e' \in E'$ becomes a successor of c_i;
- each non tick event $e' \in E'$ is removed as well as all its successors nodes. This allows to remove branches which were already possible before the occurrence of the tick.

This simplification step has to be repeated iteratively for all the tick events. We already remarked that, during each step, for $1 \leq i \leq k$, $c_i \in E$ and $c'_i \in E'$ are such that $h(c_i) = (s_i, p_i, q_i)$ and $h(c'_i) = (s_i, p'_i, q'_i)$. One can now show that we also have $p'_i = q_i + 1$ and that this remains true after some tick events have been removed. It may also be shown that the order in which tick events are considered has no influence on the final result.

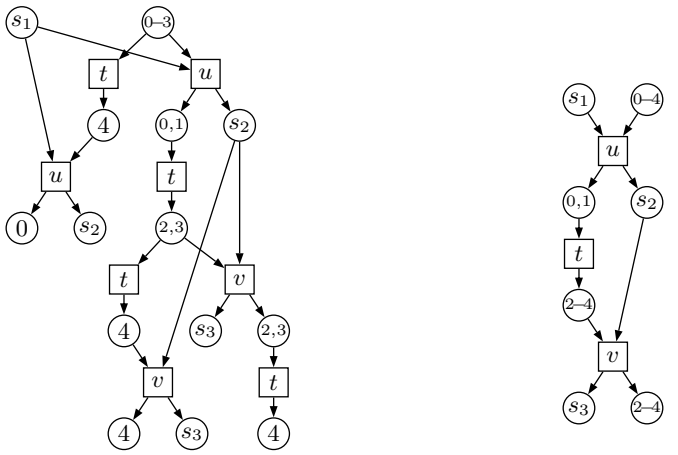

Fig. 5. On the left, the prefix generated using the first method, and on the right, using the second one. The bindings are no more relevant and thus not indicated.

By applying this transformation, we obtain the prefix given on the left of the figure 5, notice that some conditions are now labelled by lists or ranges of integers when they correspond to several possible markings of s_c. One can see that the left part of the original prefix have been simplified and that the only remaining visible tick is the one which leads to have 4 in s_c thus disabling any further tick. Similarly, the right branch was also simplified and the two remaining occurrences of v correspond to the two following situations: both v and t are enabled; or, only v is enabled.

It may be considered that too much information is still present in the prefix. In particular, one can distinguish between states from which tick can or cannot occur, which is an information only related to our particular modelling of time. In order to simplify again, we can use the same transformation scheme but, instead of removing a tick event when $\mathsf{trans}(E) = \mathsf{trans}(E')$, we use the weaker condition $\mathsf{trans}(E/\tau) = \mathsf{trans}(E'/\tau)$ where X/τ is X from which all the tick events have been removed. This new criterion leads to the prefix given on the right of the figure 5 in which the only remaining tick event denotes that time *must* pass. All the situations in which time only *may* pass have been hidden. Choosing one or the other criterion depends if one wants to always know when ticks are possible or not. But, in both cases, we achieved our goal which was to remove the sensitivity to constants.

5 Conclusion

We defined a process algebra with multiway communication and timing feature through clocks directly handled. This model, the *Causal Time Calculus* (*CTC*),

was provided with a structural operational semantics as well as with a consistent denotational semantics in terms of labelled coloured Petri nets. These nets use the so called *causal time* approach to the modelling of time which was shown in a previous paper [3] having the potentiality for efficient verification but suffering from a sensitivity to the constants compared to clocks. An important contribution of this paper was to show how to remove this weakness.

As an extension of the *Petri Box Calculus* (*PBC*) [2], CTC is similar to the approach in [11] where the author extends PBC with time using *time Petri nets* [15] for the denotational semantics. A similar result is also obtained in [14] where *timed Petri nets* [19] are used. It should be noted that, in both cases, the model checking of the underlying models is known to be much less efficient than that of standard Petri nets. This makes an importance difference with CTC for which the efficiency of the verification was a major concern. Moreover, we introduced time through explicit clocks directly handled by the processes which is known to be useful for modelling timed systems (this is indeed the scheme used in timed automata). Among process algebras not related to PBC, we should distinguish ARTS [6, chap. 5] which has been designed in order to denote timed automata while CTC denotes Petri nets; ARTS thus provides continuous time while CTC uses discrete time. Another difference is that the operational semantics in ARTS is used to give a translation from terms to automata while in CTC it is independent of Petri nets (even if both semantics are consistent). It finally appears that both algebras may be complementary as they denote objects on which model checking can be performed efficiently. Which one to use in which case is still a topic for future research. Concerning the other process algebra with time (for instance those based on CCS, see [5]), it may be remarked that most of them also use ticks to model the passing of time. However, they generally consider an interleaving semantics of parallelism while CTC considers true concurrency and most of these algebras do not provide multiway as in CTC.

Several extensions to the model presented here can be envisaged, in particular: actions with parameters, allowing to exchange data during handshakes; buffered communication, allowing to model program variables; and guards, allowing to specify conditions under which an atomic process may be executed. Incorporating these features should be straightforward since they are already defined in several extensions of PBC (in particular in [4]). Another extension would be to allow the maximum values of clocks to be changed dynamically. This must be addressed carefully in order to guaranty that either a finite prefix of the unfoldings can always be found or methods dedicated to infinite state spaces can be used.

Last but nor least, an in-depth study of the unfolding simplification proposed here appears necessary in order to know exactly what is its influence on the properties which can be verified: which one are preserved and which one are hidden. One way to reach this goal is to define a timed temporal logic in order to specify properties which could then be verified automatically. The more complete this logic will be, the more we will know about the properties preserved by our unfolding simplification.

Acknowledgement. I am very grateful to Victor Khomenko for his advice about Petri nets unfoldings.

References

1. R. Alur and D. Dill. *A theory of timed automata.* Theoretical Computer Science 126(2). Elsevier, 1994.
2. E. Best, R. Devillers and J. Hall. *The Petri Box Calculus: a new causal algebra with multilabel communication.* Advance in Petri nets 1992, LNCS 609. Springer, 1992.
3. C. Bui Thanh, H. Klaudel and F. Pommereau. *Petri nets with causal time for system verification.* MTCS'02, Electronic Notes in Theoretical Computer Sciences 68.5. Elsevier, 2002.
4. C. Bui Thanh, H. Klaudel and F. Pommereau. *Box Calculus with Coloured Buffers.* LACL Technical report, 2002. Available at `http://www.univ-paris12.fr/lacl`
5. F. Corradini, D. D'Ortenzio and P. Inverardi. *On the relationship among four timed process algebras.* Fundamenta Informaticae 34. IOS Press, 1999.
6. P.R. D'Argenio. *Algebras and automata for real-time systems.* PhD. Thesis, Department of Computer Science, University of Twente, 1999.
7. R. Durchholz. *Causality, time, and deadlines.* Data & Knowledge Engineering 6. North-Holland, 1991.
8. J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming 23. Elsevier, 1994.
9. V. Khomenko, M. Koutny et W. Vogler. *Canonical prefixes of Petri net unfoldings.* CAV'02, LNCS 2404. Springer, 2002.
10. V. Khomenko and M. Koutny. *Branching processes of high-level Petri nets.* TACAS'03, LNCS 2619. Springer, 2003.
11. M. Koutny. *A compositional model of time Petri nets.* ICATPN'00, LNCS 1825. Springer, 2000.
12. K. G. Larsen, P. Pettersson et W. Yi. *UPPAAL in a nutshell.* International Journal on Software Tools and Technology Transfer 1(1-2). Springer, 1997.
13. M. Mäkelä. *MARIA: modular reachability analyser for algebraic system nets.* Online manual, `http://www.tcs.hut.fi/maria`, 1999.
14. O. Marroquín Alonzo and D. de Frutos Escrig. *Extending the Petri Box Calculus with time.* ICATPN'01, LNCS 2075. Springer, 2001.
15. P. M. Merlin and D. J. Farber. *Recoverability of communication protocols—implications of a theoretical study.* IEEE Transaction on Communication 24. IEEE Society, 1976.
16. R. Milner *Communication and concurrency.* Prentice Hall, 1989.
17. G. D. Plotkin. *A Structural approach to Operational Semantics.* Technical Report FN-19, Computer Science Department, University of Aarhus, 1981.
18. F. Pommereau. *Causal Time Calculus.* LACL Technical report, 2002. Available at `http://www.univ-paris12.fr/lacl`
19. C. Ramchandani. *Analysis of asynchronous concurrent systems using Petri nets.* PhD. Thesis, project MAC, MAC-TR 120. MIT, 1974.
20. S. Yovine. *Kronos: A verification tool for real-time systems.* International Journal of Software Tools for Technology Transfer, 1(1/2). Springer, 1997.