Timing and Causality in Process Algebra

LUCA ACETO AND DAVID MURPHY

University of Sussex and University of Birmingham

ABSTRACT. There has been considerable controversy in concurrency theory between the 'interleaving' and 'true concurrency' schools. The former school advocates associating a transition system with a process which captures concurrent execution via the interleaving of occurrences; the latter adopts more complex semantic structures to avoid reducing concurrency to interleaving.

In this paper we show that the two approaches are not irreconcilable. We define a timed process algebra where occurrences are associated with intervals of time, and give it a transition system semantics. This semantics has many of the advantages of the interleaving approach; the algebra admits an expansion theorem, and bisimulation semantics can be used as usual. Our transition systems, however, incorporate timing information, and this enables us to express concurrency: merely adding timing appropriately generalises transition systems to asynchronous transition systems, showing that time gives a link between true concurrency and interleaving. Moreover, we can provide a complete axiomatisation of bisimulation for our algebra; a result that is often problematic in a timed setting.

Another advantage of incorporating timing information into the calculus is that it allows a particularly simple definition of action refinement; this we present. The paper concludes with a comparison of the equivalence we present with those in the literature, and an example system specification in our formalism.

1. Introduction

The process algebra community is divided between those who favour *interleaving*, and those who do not. The former school reduce a process like $a\,.\,\textit{NIL} \;||\; b\,.\,\textit{NIL}$ to the choice between the alternative interleavings $(a\,.\,b\,.\,\textit{NIL}) + (b\,.\,a\,.\,\textit{NIL})$, enabling them to associate a labeled tree with a process, while the latter use more sophisticated semantic structures, such as partial orders, to distinguish between 'true' concurrency and interleaving. These structures have the advantage of being more expressive than labeled trees, but the disadvantage of often being more complex and difficult to reason with.

The correct choice of semantic structure for concurrency becomes even more problematic when we move to timed process algebra. Many of the classic interleaving algebras, such as BPA [7], 'theoretical' CSP [31] or CCS [37] have timed analogues ([6], [16] and [40] respectively), and these inherit many of the features of the interleaved originals. However, timing adds a new element: in an untimed algebra, we would associate the traces

$$\langle a, b, c\rangle, \qquad \langle a, c, b\rangle \qquad \text{and} \qquad \langle c, a, b\rangle$$

Postal addresses: Dr. Luca Aceto, School of Cognitive and Computing Science, University of Sussex, Brighton BN1 9QH, England and Dr. David Murphy, Department of Computer Science, University of Birmingham, Birmingham B15 2TT, England.

with the process $(a \,.\, b \,.\, \textit{Nil}) \;||\; c \,.\, \textit{Nil}$. If we add timing, writing $a@t$ for an a at time t, and assume that b happens 2 time-units after a, then merely generalising the traces above gives us

$$\langle a@0, b@2, c@0\rangle, \qquad \langle a@0, c@0, b@2\rangle \qquad \text{and} \qquad \langle c@0, a@0, b@2\rangle$$

Many authors [6, 30, 40] argue that the first trace, $\langle a@0, b@2, c@0\rangle$, is *inconsistent* or *ill–timed*, since the trace order does not reflect the order given by time; they claim that we cannot observe a c at time 0 after observing a b at time 2. This position allows them to reduce the allowed traces in their timed models, but at a considerable cost: expansion theorems, which are often a great aid in manipulating expressions, establishing normal forms etc., rarely follow in this setting, as Godskesen and Larsen [25] point out. Moreover, considerable trouble needs to be taken in the semantics of the calculus to ensure that such ill–timed traces do not arise, with the result that it is often possible to write processes which do not allow time to pass. This is well known in the timed process algebra community; cf. [30, 32, 50].

Here we will argue that it is not necessary to ban ill–timed traces provided they are *well–caused*. There is no *causal* connection between b and c in the process $(a \,.\, b \,.\, \textit{Nil}) \;||\; c \,.\, \textit{Nil}$, so we claim that it is unreasonable for c to influence the presence or absence of b in a semantic structure. Thus we will allow ill–timed traces, provided they come about through parallelism.

This novel approach has a number of advantages; firstly it allows us to keep many of the advantages of the classic interleaving approach, such as the existence of an expansion theorem. Secondly, it allows us to capture concurrency information purely through timing, so that we can record the difference between interleaving and concurrency without using a semantic structure more complex than a labeled tree. Finally it allows us to define a timed process algebra that is both implementable (our calculus will suffer from no pathological processes such as those which do not allow time to pass) and fairly expressive.

We will proceed thus: firstly we define the process algebra *cIpa* (for 'closed interval process algebra') that will be our main object of study. This is a subalgebra of the second author's *Ipa* introduced in [41, 42]. An operational semantics is given to the algebra *cIpa* which allows us to associate an action-timed transition system with a process. This semantics has several novel features:

- we have *urgent* actions: things happen as soon as they can. This allows us to reason compositionally about timing properties;
- we introduce local clocks to record the evolution of different parts of a distributed state. This, together with our treatment of synchronisation (which ensures that process synchronisation implies clocks synchronisation), allows us to treat parallelism smoothly.

A congruence is then defined over the action-timed transition systems given by the operational semantics, and is axiomatised. We prove that it remains a congruence under a definition of semantic action refinement. These results rely on a careful analysis of timing information.

The paper concludes with a comparison of our notion of equivalence with others in the literature, and an example of the use of *cIpa*. We aim to show the usefulness of timed process algebra not just as specification calculi for timed systems, but also as possessing inherent semantic advantages over untimed calculi. Timing, for instance, increases the expressiveness of the calculus without going beyond an interleaving setting. It also simplifies the definition of action refinement, as we show.

An extended abstract of the matter presented here appears in [3].

2. A Timed Process Algebra

In this section we define a timed process algebra, giving its syntax, investigating its timing properties, and presenting an operational semantics. We show that our semantics captures concurrency information in a natural way.

2.1. Syntax

We will use a syntax derived from both CSP [31] and CCS [37], assuming as given some set of actions, A, with τ (which will be used for internal actions of a process) not in A. We also assume a bijection $\overline{} : A \rightarrow \overline{A}$ (giving the complementary action of a), extend $\overline{}$ such that $x = \overline{\overline{x}}$ for all $x \in A \cup \overline{A}$, and write $\textit{Act}$ for $A \cup \overline{A}$. We will use a, b, c, d etc. to range over $\textit{Act}$, and write $\mathbb{R}^+$ for the positive reals. The syntax of *cIpa* is then defined by

$$P \ ::= \ a\,.\,P \mid \textit{Wait}\ t\,.\,P \mid P + P \mid P \backslash C \mid \textit{Nil} \mid P \,||\, P$$

with a ranging over $\textit{Act}$, $t \in \mathbb{R}^+$, and $C \subseteq A$.

The set of all processes generated by this syntax over a fixed set of actions A will be written *cIpa*. The operators are intended to have the following informal meaning:

$a\,.\,P$	An action $a \in \textit{Act}$ followed by the process P.
$\textit{Wait}\ t\,.\,P$	A process that can do nothing but wait for time t and then become P.
$P + Q$	This is the choice between P and Q; $P + Q$ can perform either an action from P, in which case it behaves like the rest of P, or one from Q, in which case the remainder of Q follows.
$P \backslash C$	This restriction operator allows us to force some of P's actions not to occur; all of the actions in the set C are prohibited.
$\textit{Nil}$	This is the empty process which does nothing.
$P \,\|\, Q$	This is parallel composition. Each process is allowed to proceed asynchronously, with synchronisations between a and $\overline{a}$ sometimes being possible; see subsection 2.6 for details.

We will often omit the terminal *Nil*, writing for instance $a\,.\,b$ for $a\,.(b\,.\,\textit{Nil})$. Moreover, we assume that . binds more strongly than $+$ or $||$, so that $a\,.\,b + c$ should be read as the process $(a\,.\,b\,.\,\textit{Nil}) + c\,.\,\textit{Nil}$.

2.2. Duration and Nonatomicity

Consider the process $a\,.\,b$. In any realistic system there must be a delay between a and b. Clearly we have two alternatives; either to imbue operators with delay, so that a and b are atomic and time passes 'between' them, or to view actions as compound happenings having duration. We shall take the latter course, giving a function $\Delta : A \rightarrow \mathbb{R}^+$ which assigns durations to actions. The duration $\Delta(a)$ for any $a \in A$ will be assumed to be nonzero and, for technical convenience, constant over all occurrences of a. We extend Δ to $\textit{Act}$ by defining $\Delta(\overline{a}) = \Delta(a)$.

Our approach of giving durations to actions is novel but not entirely new; Lamport assumes nonatomicity [34] in his discussions of distributed systems, as do Best and Koutny [10] when discussing priority. Related process algebraic work with explicit starts and finishes is due to Hennessy and the first author [2, 29], while the second author's [43] treats a partial order model with duration. van Glabbeek (in his definition of ST–bisimulation [20, 22]), Gorrieri [26] and especially Vogler [48, 49] consider durational information is necessary in the consideration of action refinement and hence in properly structuring the descriptions of systems, a view we concur with.

2.3. Urgency and Timing Properties

A key notion in *cIpa* is that of *urgency*; an action happens as soon as possible. Thus if a has duration π, the P in $a\,.\,P$ will definitely start executing exactly $3.141\ldots$ units of time after a has started.[*] The urgency of *cIpa* allows us to define the set of all times a process without

---

[*] Note that the issue of urgency is different from that of *liveness*. The term 'maximal liveness', which intuitively should mean things happening as quickly as possible, is in fact used in much of the literature

restriction can last, $\mathbf{dur}(P) \in \wp_{\mathrm{fin}}(\mathbb{R}^+ \cup \{0\})$. (Here $\wp_{\mathrm{fin}}(X)$ is the set of finite subsets of X.)

$$\begin{array}{rclrcl} \mathbf{dur}(\textit{Nil}) & = & \{0\} & \mathbf{dur}(\textit{Wait}\; t\,.\,P) & = & \{t' + t \mid t' \in \mathbf{dur}(P)\} \\ \mathbf{dur}(P + Q) & = & \mathbf{dur}(P) \cup \mathbf{dur}(Q) & \mathbf{dur}(a\,.\,P) & = & \{t' + \Delta(a) \mid t' \in \mathbf{dur}(P)\} \end{array}$$

$$\mathbf{dur}(P \parallel Q) = \{\mathbf{max}(t, t') \mid t \in \mathbf{dur}(P), t' \in \mathbf{dur}(Q)\}$$

Note that $\mathbf{dur}(P)$ for a *cIPA* process P, if it is defined, is indeed a finite set.

The existence of this function means that the restriction-free sublanguage of *cIPA* has compositional timing properties: we can reason about the timing of a *cIPA* process given knowledge of those of its subcomponents and how they are combined.

2.4. Action-Timed Transition Systems

We shall present the operational semantics of *cIPA* in the usual SOS style [45], associating a rooted transition system (or *process graph*) $(S, E, \rightarrow, s_0)$ with a *cIPA* process. This transition system represents the possible steps in executing P, so S will be the set of reachable configurations of P, E a set of actions, $\rightarrow$ an evolution or transition relation (which indicates which action is associated with a given state change), and $s_0 \in S$ the starting configuration. In what follows, μ will range over $\textit{Act} \cup \{\tau\}$.

Timed process algebra usually assumes the existence of an observer's clock. This is a *conceptual* clock which accompanies an observer of the system, and is used to time–stamp observations; its presence does not necessarily constitute a global clocks assumption. Indeed, we shall show in this paper how to give an operational semantics for *cIPA* in which different parallel sub-processes have an independent *local* clock which they use to time–stamp observations.

To introduce time–stamps, we need to generalise the rules of operational semantics slightly. A conventional untimed transition rule takes the form opposite

$$\frac{s_i \xrightarrow{\mu_i} s_i' \qquad (i \in I)}{u \xrightarrow{\mu} u'}$$

indicating that if every configuration s_i can evolve by μ_i to s_i' then u can evolve by μ to u'. We want to capture both the timing of an action ($\mu@t$, read 'μ occurs at time t') and its duration (δ), so we shall write timed transition rules in the form opposite

$$\frac{s_i \xrightarrow[\delta_i]{\mu_i@t_i} s_i' \qquad (i \in I)}{u \xrightarrow[\delta]{\mu@t} u'}$$

Definition 1. An *action-timed transition system* or *Atts* is a tuple $G = (S, E, \rightarrow, s_0)$ consisting of a set of configurations S, a set of timed, durationful events $E \subset (\textit{Act} \cup \{\tau\}) \times (\mathbb{R}^+ \cup \{0\}) \times \mathbb{R}^+$ (together with a duration map $\Delta : \textit{Act} \rightarrow \mathbb{R}^+$), a relation $\rightarrow\, : S \times E \times S$ and an initial configuration s_0. We usually write $s \xrightarrow[\delta]{\mu@t} u$ rather than $(s, (\mu, t, \delta), u) \in \rightarrow$, and require that τ is the only action whose duration can vary

$$s \xrightarrow[\delta]{\mu@t} u \quad \text{and} \quad s' \xrightarrow[\delta']{\mu@t'} u' \qquad \text{implies} \qquad (\delta = \delta' \;\vee\; \mu = \tau)$$

For the sake of convenience, we shall only consider *Atts*s with the following properties:

lack of cycles Let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$. Then we require:

$$s \xrightarrow{e} u \qquad \text{implies} \qquad u \not\rightarrow^* s$$

---

to mean τ actions happening as quickly as possible. In the absence of duration information, this can cause effects, such as the a in $\tau\,.\,a$ happening immediately, which are not present in *cIPA*. As illustration, consider $(a\,.\,b \parallel \overline{a})\backslash\{a\}$; in *cIPA* the b will happen at time $\Delta(a)$ after the process has begun, not immediately. We take the view that there is no reason for hidden actions to lose their durations.

reachability All states mentioned are reachable from the initial configuration, and all events label some transition:

$$s \in S \qquad \text{implies} \qquad s_0 \rightarrow^* s$$

$$\text{and } e \in E \qquad \text{implies} \qquad \exists s, u \in S \quad \text{such that} \quad s \xrightarrow{e} u$$

Note that the *ATTS*s we consider are *root unwound* in the sense of [20, chapter 3], i.e. they have no incoming edges at the root:

$$\neg(s \xrightarrow[\delta]{\mu@t} s_0)$$

2.5. Configurations

Consider the general timed transition rule of the last section. In keeping with the idea of urgency, we want P to begin as soon as a has ended in $a\,.\,P$, i.e. at time $\Delta(a)$ assuming that $a\,.\,P$ started at time 0. However, if configurations are just process fragments, we have no convenient syntactic means of knowing the time of occurrence of an action. Thus we define the set $\mathcal{C}(\textit{cIPA})$ of *configurations* as follows:

$$s \;::=\; P\,t \;\mid\; s + s \;\mid\; s \,||\, s \;\mid\; s\backslash C$$

The idea is that $P\,t$ is the process P started at time t; we conventionally identify P with $P\,0$. (Note that, with this convention, we can regard *cIPA* as a sublanguage of $\mathcal{C}(\textit{cIPA})$,— indeed we have identified *cIPA* with a subset of the set of *generators* of $\mathcal{C}(\textit{cIPA})$.) The configuration $(P\;t)\;||\;(P'\;t')$ is P with local clock at t in parallel with P' whose local clock reads t'. This use of local clocks enables us to treat the simultaneous execution of actions in parallel processes smoothly. It is important to emphasise here that our local clocks are conceptual; they are an aid in recording distributed state, rather than a precise model of an implementation.

For the sake of simplicity, we shall follow the example of [15] and consider configurations in canonical form with respect to the operators $\_\;t$. In particular, it will be assumed that the operators $\_\;t$ distribute over nondeterministic choice, restriction and parallel composition. Formally, let $\equiv$ denote the least congruence over these operators which satisfies the following axioms:

$$\begin{array}{rcl} (P + Q)\,t & = & P\,t + Q\,t \\ (P\backslash C)\,t & = & (P\,t)\backslash C \\ (P \,||\, Q)\,t & = & (P\,t) \,||\, (Q\,t) \end{array}$$

Applying these axioms as rewrite rules from left to right, it is easy to see that for each $s \in \mathcal{C}(\textit{cIPA})$ there exists a *canonical term* generated by the grammar

$$s \;::=\; \textit{Nil}\,t \;\mid\; (a\,.\,P)\,t \;\mid\; (\textit{Wait}\;t'\,.\,P)\,t \;\mid\; s + s \;\mid\; s\backslash C \;\mid\; s \,||\, s$$

such that s is always $\equiv$ to a canonical term. In what follows, $\mathcal{C}(\textit{cIPA})$ will always be considered modulo $\equiv$.

Notice, incidentally, that there are some configurations that do not correspond to processes in any recognisable sense; $(a\;2) + (b\;3)$, for instance, is not an implementable choice. Such configurations are needed for semantic reasons, (allowing us to show completeness, for instance) and should not be thought as corresponding to (the state of) a process which offers such a choice.

We shall write transition rules between configurations, a typical example being

$$s \xrightarrow[\delta]{\mu@t} s'$$

which means that starting from the configuration s, the configuration s' is reachable by the action μ happening at time t with duration δ.

Definition 2 (Internal transition relation). In a given *ATTS*, $(S, E, \rightarrow, s_0)$, we say that there is an internal transition from $s \in S$ to $u \in S$, written $s \Rightarrow u$, iff there is a sequence of τ–transitions $s \xrightarrow[\delta_1]{\tau@t_1} \cdots \xrightarrow[\delta_n]{\tau@t_n} u$.

Note that we do not associate a duration with internal transitions. The reason for this choice is essentially technical. The notion of behavioural equivalence we shall impose on *cIPA* processes will, to a certain extent, abstract from their internal evolution, and all the information about the changes in the local clocks of processes which occur in the transition $s \Rightarrow u$ will be recorded in the target configuration u—ready to be used to associate the appropriate time-stamp with the next observable action.

2.6. *ATTS* Semantics

In this subsection, we shall define the *ATTS* associated with a *cIPA* expression; see displays 1 and 2. Our timed semantics will be rather different from the usual ones in the literature [6, 30, 40, 47] in that we do not explicitly allow time to pass. Rather, we merely say when actions happen. This means that processes will not be able to refuse to let time pass. Moreover, our association of a non–zero duration with each action means that we cannot build Zeno machines—machines that do more than a finite number of actions in a finite interval of time. These two features mean that all processes in our calculus will be *implementable* [33, 44].

A few comments on the rules in displays 1 and 2 are now in order. First primitive happenings; the rule ACT. Actions fire as soon as they are ready, passing control to their suffixes once their duration is over.

Waits (rule WAIT) just wait the specified time with a τ transition, passing control to their suffixes at the end of their wait. We want to think of *WAIT* $t\,.\,P$ as being a timed version of CCS's $\tau.P$ rather than timed CSP's `wait` $t \rightarrow P$. This is because we think the timing of choices is important; we want to make a distinction between a choice followed by a wait and a wait followed by a choice, i.e. between

$$(\mathit{WAIT}\ 2\,.\,P\ +\ \mathit{WAIT}\ 2\,.\,Q) \qquad \text{and} \qquad \mathit{WAIT}\ 2\,.(P\ +\ Q)$$

This is just the timed version of the distinction between $\tau.P + \tau.Q$ and $\tau.(P+Q)$. (In contrast, some algebras, such as Hennessy and Regan's [30], do not allow 'the passage of time to decide a choice', and hence have an equality between wait-then-choice and choice-then-wait situations, which means that moments of choice cannot be clearly distinguished.)

$$\textsc{Act}\ \frac{}{(a\,.\,P)\ t \xrightarrow[\Delta(a)]{a@t} P\ (t+\Delta(a))} \qquad\qquad \textsc{Wait}\ \frac{}{(\mathit{WAIT}\ t'\,.\,P)\ t \xrightarrow[t']{\tau@t} P\ (t+t')}$$

$$\textsc{Res}\ \frac{s \xrightarrow[\delta]{a@t} s'}{(s \backslash C) \xrightarrow[\delta]{a@t} (s' \backslash C)}\ a, \overline{a} \notin C$$

$$\text{CL}\ \frac{s_1 \xrightarrow[\delta]{\mu@t} s_1{}'}{s_1 + s_2 \xrightarrow[\delta]{\mu@t} s_1{}'} \qquad\qquad \text{CR}\ \frac{s_1 \xrightarrow[\delta]{\mu@t} s_1{}'}{s_2 + s_1 \xrightarrow[\delta]{\mu@t} s_1{}'}$$

DISPLAY 1. The timed operational semantics for *cIPA* without parallelism.

The rule for restriction is RES : restriction just stops the restricted actions from happening; τs can always proceed since $C \subseteq A$, and $\tau \notin A$.

The rules for parallelism are PAL, PAR and SYNC. The idea of a configuration $P\,t \mid\mid Q\,t'$ is that t and t' are the local clocks at P and Q. Notice that the two clocks never interact during

$$\text{PAL}\ \frac{s_1 \xrightarrow[\delta]{\mu @ t} s_1{}'}{s_1 \mid\mid s_2 \xrightarrow[\delta]{\mu @ t} s_1{}' \mid\mid s_2} \qquad \text{PAR}\ \frac{s_1 \xrightarrow[\delta]{\mu @ t} s_1{}'}{s_2 \mid\mid s_1 \xrightarrow[\delta]{\mu @ t} s_2 \mid\mid s_1{}'}$$

$$\textsc{Sync}\ \frac{s_1 \xrightarrow[\delta]{a @ t} s_1' \quad s_2 \xrightarrow[\delta]{\overline{a} @ t} s_2'}{s_1 \mid\mid s_2 \xrightarrow[\delta]{\tau @ t} s_1' \mid\mid s_2'}$$

Display 2. The timed operational semantics of parallelism.

the execution of $P \mid\mid Q$, except during synchronisation, so they are genuinely local. Notice too that synchronisation is only allowed provided the local clocks of the synchronising actions match exactly (rule Sync). This seems reasonable, as any protocol which implements synchronisation requires a set-up phase before the synchronisation can be said to have happened, during which what effectively happens is a synchronisation of clocks [4]. Moreover, our clocks are not supposed to model all of the features of physical clocks in actual implementations, so abstracting away from the details of clock synchronisation is not unreasonable. Note, though, that our assignment of clocks to each parallel process is rather close to Matthern's notion of vector time [35] in which setting a comprehensive account of Lamport's clock synchronisation procedure can be given [34].

The choice of allowing synchronisation only if the local clocks of the synchronising processes match exactly, however, is not without consequences. Consider the process $(a \,.\, b \mid\mid \overline{b} \,.\, c)\backslash\{b\}$. In the presence of rule Sync, this process will perform an a-action at time 0 and then deadlock, because $b@\Delta(a)$ and $\overline{b}@0$ cannot synchronise. The point here is that achieving synchronisation is a responsibility of the implementer of a system, not an automatic right; one has to insert waits in processes to make sure that the desired synchronisations take place. It is this feature of low–level implementations (rather than higher–level descriptions of them) that we model here.

3. The Meanings of Processes

The rules of the last section allow us to associate an *Atts* with a configuration of a *cIpa* process. Here we identify the meaning of a process P as the *Atts* generated by the configuration $P\,0$, and present a suitable notion of equivalence over such transition systems. This equivalence is shown, through an analysis of its interaction with timing information, to be a congruence. This analysis is then extended, showing how timing captures concurrency information.

Definition 3. We will write $[\![P]\!]$ for the process graph $G = (S, E, \rightarrow, s_0)$ generated by a *cIpa* process P starting at time 0 from the rules in displays 1 and 2.

3.1. Rooted Branching Bisimulation

We want to define a notion of equivalence over *Atts*s that will tell us when two *cIpa* processes should be regarded as equal. The notion we pick should reflect our interest in the precise timing of choices discussed above, and thus should respect the branching structure of processes. Given this preoccupation, we shall pick the notion of branching bisimulation introduced by van Glabbeek and Weijland [23]. This equivalence is stronger than weak bisimulation, and accurately captures branching structure.

Definition 4. A *timed branching bisimulation* from an *Atts* $G = (S, E, \rightarrow, s_0)$ to another $H = (S', E', \rightarrow', s_0')$ is a symmetric relation $R : (S \times S') \cup (S' \times S)$ such that

(i) The roots are related; $s_0 \; R \; s_0'$.

(ii) If two configurations are related and an observable timed transition with a given duration is possible from one of them, then it is possible from the other with the same time and duration: if we have $s \; R \; u$ and $s \xrightarrow[\delta]{\mu @ t} s'$ then exactly one of the following applies

$\mu = \tau$. In this case we require that either $s' \; R \; u$ or $u \Rightarrow u' \xrightarrow[\delta']{\tau @ t'} u''$, for some u', u'' such that $s \; R \; u'$ and $s' \; R \; u''$.

$\mu = a$. Here we require that $u \Rightarrow u' \xrightarrow[\delta]{a @ t} u''$ for some u', u'' such that $s \; R \; u'$ and $s' \; R \; u''$.

If there is a timed branching bisimulation between G and H, we say that that G and H are branching bisimilar. If R is a branching bisimulation and additionally it only relates root nodes to root nodes, we will call it 'rooted'. Since 'rooted branching bisimulation' is a somewhat cumbersome term, we will abbreviate such relations by 'RBB's. We write $R : G \approx H$ to indicate that G and H are rooted branching bisimilar, and that this fact is witnessed by the relation R; the R is sometimes dropped. Finally, two *cIPA* processes P and Q are rooted branching bisimilar iff $[\![P]\!]$ and $[\![Q]\!]$ are.

The reader will have noticed that, in the definition of timed branching bisimulation, we abstract from the duration and start time of internal transitions. This is in agreement with Definition 2. However, as previously remarked, the duration of internal transitions is not simply forgotten. In fact, internal transitions change the local clocks in a configuration, and so their duration may influence the start time of observable transitions and hence bisimilarity: in the definition of bisimulation we are only allowed to match observable actions which have the same start time. We return to this point in subsection 3.2.

Examples. To provide some intuition, we give the meanings of some processes, and present some equations between processes which do and do not hold relative to RBB. As usual, we write $P = Q$ for $[\![P]\!] \approx [\![Q]\!]$.

Interleaving Consider $a \,||\, b$. This generates the traditional diamond with paths

$$(a \,||\, b)\, 0 \;\xrightarrow[\Delta(a)]{a@0}\; (\textit{Nil}\,\Delta(a)) \,||\, (b\, 0) \;\xrightarrow[\Delta(b)]{b@0}\; (\textit{Nil}\,\Delta(a)) \,||\, (\textit{Nil}\,\Delta(b))$$

and

$$(a \,||\, b)\, 0 \;\xrightarrow[\Delta(b)]{b@0}\; (a\, 0) \,||\, (\textit{Nil}\,\Delta(b)) \;\xrightarrow[\Delta(a)]{a@0}\; (\textit{Nil}\,\Delta(a)) \,||\, (\textit{Nil}\,\Delta(b))$$

In contrast,

$$(a \,.\, b \;+\; b \,.\, a)\, 0 \;\xrightarrow[\Delta(a)]{a@0}\; b\,\Delta(a) \;\xrightarrow[\Delta(b)]{b@\Delta(a)}\; \textit{Nil}\,(\Delta(a) + \Delta(b))$$

and

$$(a \,.\, b \;+\; b \,.\, a)\, 0 \;\xrightarrow[\Delta(b)]{b@0}\; a\,\Delta(b) \;\xrightarrow[\Delta(a)]{a@\Delta(b)}\; \textit{Nil}\,(\Delta(a) + \Delta(b))$$

clearly indicating the use of local clocks in differentiating between these two processes.

Restriction Consider $P = (a \,.\, b \,||\, \overline{a})\backslash\{a\}$ and $Q = (c \,.\, b \,||\, \overline{c})\backslash\{c\}$. Then it is easy to see that $P \approx Q$ iff $\Delta(a) = \Delta(c)$ despite the fact that both processes are somehow semantically sequential. This shows that, in general, RBB is even finer than strong bisimulation over semantically finite sequential processes, which is not surprising given the rôle played by timing information in *cIPA*. Another similar example is $(a \,.\, c \,.\, b \,||\, \textit{Wait}\ \Delta(a) \,.\, \overline{c})\backslash\{c\}$ and $(a \,.\, d \,.\, b \,||\, \textit{Wait}\ \Delta(a) \,.\, \overline{d})\backslash\{d\}$; two processes which are only equated by our theory if $\Delta(c) = \Delta(d)$.

Law 1 The usual equations $P + P = P$ and $P + \textit{Nil} = P$ hold for our notion of equivalence.

NonLaw It is interesting that none of the τ laws in [23] immediately generalises to our setting. For instance, the restriction example above generalises to $a \,.\, \textit{Wait}\, t \,.\, b \not\approx a \,.\, b$. We also have $\textit{Wait}\, t \,.\, a \not\approx \textit{Wait}\, t \,.\, a + a$ since the latter can begin a earlier than the former.

Note, however, that $a\,.\,\textsc{Wait}\;t \;\approx\; a$ as the only behaviour that can be observed of both processes is that they can execute action a at time 0 with duration $\Delta(a)$.

Law 2 There are some nontrivial relationships between processes with τs, e.g.

$$a\,.\,\textsc{Wait}\;t\,.\,\textsc{Wait}\;t'\,.\,P \;=\; a\,.\,\textsc{Wait}\;(t+t')\,.\,P$$

Indeed, it is easy to see that $a\,.\,\textsc{Wait}\;t \;\approx\; a\,.\,\textsc{Wait}\;t'$ and $\textsc{Wait}\;t \;\approx\; \textsc{Wait}\;t'$ for all $t, t' \in \mathbb{R}^+$. Moreover, we have that $a\,.(\textsc{Wait}\;t \,||\, b) \;\approx\; a\,.\,b$.

3.2. Compositionality and Timing

We will now show that rooted branching bisimulation is a congruence, and discuss the timing–dependence of our results. More precisely, we relate the compositionally defined duration function $\mathbf{dur}(P)$ (which was defined for restriction-free processes in section 2.3) and the operational semantics given above, showing that for each such process, $t \in \mathbf{dur}(P)$ iff the configuration $P\,0$ can evolve to a terminal configuration s with maximum time t according to the operational semantics. This then allows us to prove the compositionality result.

Definition 5. Let $\mathbf{maxtime}(s)$ denote the largest clock time t occurring in s, and, for each $t \geq 0$, $s\nearrow^t$ denote the configuration obtained by adding t to each clock time occurring in s, i.e. $\_\nearrow^t$ is the unique homomorphism satisfying

$$\begin{aligned}
\textsc{Nil}\;t'\nearrow^t &= \textsc{Nil}\;(t'+t) \\
(a\,.\,P)\;t'\nearrow^t &= (a\,.\,P)\;(t'+t) \\
(\textsc{Wait}\;t''\,.\,P)\;t'\nearrow^t &= (\textsc{Wait}\;t''\,.\,P)\;(t'+t)
\end{aligned}$$

The following lemma can then be easily shown by structural induction on s:

Lemma 6. For all $s \in \mathcal{C}(\textsc{cIpa})$ and $t \in \mathbb{R}^+$, the following statements hold:

(i) $s \xrightarrow[\delta]{\mu @ t'} s'$ implies $s\nearrow^t \xrightarrow[\delta]{\mu @ (t'+t)} s'\nearrow^t$;

(ii) $s\nearrow^t \xrightarrow[\delta]{\mu @ t_1} s_1$ implies $s \xrightarrow[\delta]{\mu @ t'} s'$ for some $s' \in \mathcal{C}(\textsc{cIpa})$ and $t' \in \mathbb{R}^+$ such that $s_1 = s'\nearrow^t$ and $t_1 = t' + t$;

(iii) $\mathbf{maxtime}(s\nearrow^t) = t + \mathbf{maxtime}(s)$.

This gives us that the function **dur** is indeed in agreement with the operational semantics:

Proposition 7. For all restriction-free $\textsc{cIpa}$ processes P and times $t \in \mathbb{R}^+$, $t \in \mathbf{dur}(P)$ iff there exists $s \in \mathcal{C}(\textsc{cIpa})$ such that $P \longrightarrow^\star s \not\longrightarrow$ and $\mathbf{maxtime}(s) = t$.

Proof. A straightforward induction on the structure of P, using Lemma 6 in the cases dealing with action and $\textsc{Wait}\;t$ prefixing. ☐

We can now tackle the compositionality of $\textsc{Rbb}$:

Theorem 8. Rooted branching bisimulation is compositional; $[\![P]\!] \;\approx\; [\![Q]\!]$ implies

$$[\![P \ominus R]\!] \approx [\![Q \ominus R]\!]$$

for $\ominus \in \{+, ||\}$. Furthermore, for any $t \in \mathbb{R}^+$ and $a \in \textsc{Act}$,

$$[\![a\,.\,P]\!] \;\approx\; [\![a\,.\,Q]\!], \qquad [\![\textsc{Wait}\;t\,.\,P]\!] \;\approx\; [\![\textsc{Wait}\;t\,.\,Q]\!] \qquad \text{and} \qquad [\![P\backslash C]\!] \;\approx\; [\![Q\backslash C]\!]$$

Proof. This is an extension to $\textsc{cIpa}$ of van Glabbeek and Weijland's for BPA [23]. The only cases which depart slightly from the standard proof are those for action and $\textsc{Wait}\,t$ prefixing. To show that $\approx$ is preserved by these operations, we prove that if $R : [\![P]\!] \;\approx\; [\![Q]\!]$, then the symmetric closure of the relations

$$\begin{aligned}
R_{pre} &= \{(a\,.\,P, a\,.\,Q)\} \cup \{(s_1\nearrow^{\Delta(a)}, s_2\nearrow^{\Delta(a)}) \mid s_1 \; R \; s_2\} \\
R_{wait} &= \{(\textsc{Wait}\;t\,.\,P, \textsc{Wait}\;t\,.\,Q)\} \cup \{(s_1\nearrow^{t}, s_2\nearrow^{t}) \mid s_1 \; R \; s_2\}
\end{aligned}$$

are rooted branching bisimulations between the relevant process graphs. This can be easily verified using Lemma 6. □

3.3. Time Uniformity

The reals are often problematic in computer science because they admit unrealisable features: the function that takes value a at the rationals and b at the irrationals, for instance, does not correspond to the timed execution of a process we can build. In the last subsection we showed that our semantics is *uniformly parameterised* by the reals; lemma 6 states that we can add t to all the clocks in any configuration, and the effect will just be to shift all of the transitions from that configuration forward t in time. Thus our semantics varies smoothly rather than pathologically with time.

Here we examine another facet of the interaction of timing and behaviour —how $\approx$ varies as the duration function is changed. Clearly, the identifications made by the congruence $\approx$ depend on the particular choice of Δ. For instance, an equality like

$$(a\,.\,b \,||\, \overline{a})\backslash a \;\approx\; \textit{W}\textsc{ait}\, t\,.\,b$$

holds iff $\Delta(a) = t$.

If for all a, $\Delta(a) = \delta$ for some $\delta > 0$, we recover some of the identifications of an untimed equivalence from $\approx$; here, for instance, we have

$$(a\,.\,b \,||\, \overline{a})\backslash a \;\approx\; (c\,.\,b \,||\, \overline{c})\backslash c$$

However, the presence of timing information still allows us to make distinctions which are not made by untimed equivalences. For example, we would still differentiate the processes $(a\,.\,b \,||\, \overline{a})\backslash a$ and $(a\,.\,a\,.\,b \,||\, \overline{a}\,.\,\overline{a})\backslash a$, which are identified by all the untimed equivalences which abstract from the internal evolution of processes we are aware of.

We shall now show that, if we restrict ourselves to constant duration functions, $\approx$ does not depend on the choice of duration for the actions over *Wait* t–free processes. First, we introduce some notation that will be useful in the proof of this result. For each $\delta \in \mathbb{R}^+$, we shall write $\approx_\delta$ for the rooted branching bisimulation over *Wait* t–free processes induced by the duration function which assigns duration δ to each action; using such a duration function, all the transitions between *Wait* t–free configurations will have duration δ. Furthermore, we write $\xrightarrow[\delta]{\mu@t}$ for the μ-transition relation induced by such a duration function, and $[\![P]\!]_\delta$ for the *Atts* associated with a process P by the operational semantics in this setting.

Definition 9. Let $\delta \in \mathbb{R}^+$. A *cIPA* configuration s is *consistent* with δ iff for each clock t occurring in s, $t = n\delta$ for some non-negative integer n.

An induction on the depth of the proof of the relevant transition now shows that:

Lemma 10. Let s be a *Wait* t–free *cIPA* configuration which is consistent with δ. Suppose that $\Delta(a) = \delta$ for each action a and that $s \xrightarrow[\delta]{\mu@t} s'$. Then $t = n\delta$ for some non-negative integer n, and s' is a *Wait* t–free *cIPA* configuration which is consistent with δ.

As an immediate corollary of the above lemma, we have that, for each *Wait* t–free *cIPA* process P, each state s in the *Atts* $[\![P]\!]_\delta$ is a *cIPA* configuration which is consistent with δ.

Corollary 11. Let P be a *Wait* t–free *cIPA* process and assume that $\Delta(a) = \delta$ for each action a. Then $P\,0 \rightarrow^{\star} s$ implies that s is a *Wait* t–free *cIPA* configuration which is consistent with δ.

For each *cIPA* configuration s which is consistent with δ, we define its translation to a time scale with unit of measure $\delta' \in \mathbb{R}^+$ as the configuration $s[\delta \mapsto \delta']$ obtained by changing each clock $t = n\delta$ in s to $t' = n\delta'$. Note that $(P\,0)[\delta \mapsto \delta'] = P\,0$ for each *cIPA* process P and $\delta' \in \mathbb{R}^+$.

The following important lemma relates the operational semantics of a *Wait* t–free *cIPA* configuration which is consistent with δ with that of its translation in time $s[\delta \mapsto \delta']$.

Lemma 12. Let s be a *Wait* t–free configuration which is consistent with δ, and let $\delta' \in \mathbb{R}^+$. Then the following statements hold:

(i) $s \xrightarrow[\delta]{\mu @ (n\delta)} s'$ implies $s[\delta \mapsto \delta'] \xrightarrow[\delta']{\mu @ (n\delta')} s'[\delta \mapsto \delta']$ and

(ii) $s[\delta \mapsto \delta'] \xrightarrow[\delta']{\mu @ t} s''$ implies $t = n\delta'$ for some non-negative integer n, and $s \xrightarrow[\delta]{\mu @ (n\delta)} s'$ for some s' such that $s'' = s'[\delta \mapsto \delta']$.

We are now in a position to prove that Rbb is independent of the choice of a constant duration function over *Wait* t-free processes.

Proposition 13. For all *Wait* t-free *cIpa* processes P and Q, and positive reals δ and δ', $P \approx_\delta Q$ iff $P \approx_{\delta'} Q$.

Proof. By symmetry, it is sufficient to prove that $P \approx_\delta Q$ implies $P \approx_{\delta'} Q$. Assume then that $P \approx_\delta Q$ and that P and Q are *Wait* t-free *cIpa* processes. Then there exists a rooted branching bisimulation R between $[\![P]\!]_\delta$ and $[\![Q]\!]_\delta$. Our aim is to use R to define a Rbb between $[\![P]\!]_{\delta'}$ and $[\![Q]\!]_{\delta'}$ proving $P \approx_{\delta'} Q$. Consider the relation R' given by

$$R' = \{(s_1[\delta \mapsto \delta'], s_2[\delta \mapsto \delta']) \mid s_1 \; R \; s_2\}$$

First of all, note that $P\,0 \;R'\; Q\,0$ as $P\,0 \;R\; Q\,0$. To establish the claim it is thus sufficient to prove that R' is a branching bisimulation between $[\![P]\!]_{\delta'}$ and $[\![Q]\!]_{\delta'}$, which follows from Corollary 11 and Lemma 12. □

Thus, all of the $\approx_\delta$s coincide, and it is this equivalence that should be thought of as the untimed version of $\approx$. (In fact, our results carry over with little modification to processes with (suitably translated) *Wait* δs, so the presence of τs does not alter the results of this section.)

3.4. Ill–Timed Paths

Suppose that we have the following path in an *Atts*, G:

$$s \xrightarrow[\delta_1]{\mu_1 @ t_1} s' \xrightarrow[\delta_2]{\mu_2 @ t_2} s''$$

Since our actions are urgent,— they happen as quickly as they can,— we expect μ_2 to have started immediately after μ_1 has ended, and so we have

$$t_2 = t_1 + \delta_1$$

Any path which does not obey this property will be called *ill–timed*; note that ill–timed paths include those that have gaps in their time ($t_2 > t_1 + \delta_1$) and those that run backwards in time ($t_2 < t_1 + \delta_1$).

Before we analyse how ill–timed paths occur, we need a little more notation. We will introduce the non-negative integer **clocks**(s) for a configuration s, which will be the maximum number of local clocks of parallel processes which can be initially active in s.

Definition 14. Define by structural induction on (the canonical form of) configurations, s,

$$\begin{array}{rcl}
\mathbf{clocks}(\mathit{Nil}\; t) & = & 0 \\
\mathbf{clocks}((a\,.\,P)\; t) & = & 1 \\
\mathbf{clocks}((\mathit{Wait}\; t'\,.\,P)\; t) & = & 1 \\
\mathbf{clocks}(s + s') & = & \mathbf{max}(\mathbf{clocks}(s), \mathbf{clocks}(s')) \\
\mathbf{clocks}(s \mid\mid s') & = & \mathbf{clocks}(s) + \mathbf{clocks}(s') \\
\mathbf{clocks}(s \backslash C) & = & \mathbf{clocks}(s)
\end{array}$$

Next, an important lemma on the clocks in configurations;

Lemma 15 (The Clocks Lemma). Suppose that $s \xrightarrow[\delta]{\mu @ t} s'$ in $[\![P]\!]$ for some process P. Then there is a clock in s reading t and $t + \delta$ in s'.

Proof. By induction over the rules in displays 1 and 2 given above. ☐

Our next proposition shows that ill–timed paths precisely arise through concurrency:

Proposition 16. Suppose that P is a *cIPA* process without restriction. Then there is a configuration s in $[\![P]\!]$ with $\mathbf{clocks}(s) > 1$ if and only if $[\![P]\!]$ contains an ill–timed path.

Proof. We first prove the 'if' direction of the statement. Suppose that $P\,0 \to^\star s$ and $s \xrightarrow[\delta_1]{\mu_1 @ t_1} s' \xrightarrow[\delta_2]{\mu_2 @ t_2} s''$ is ill–timed. Then, by the clocks lemma, there are clocks in s and s' reading t_1 and t_2 respectively. But $t_1 + \delta \neq t_2$, hence, by the clocks lemma, s' has at least one clock reading $t_1 + \delta$ and one reading t_2. However, inspection of the only clock introduction rules (PAL and PAR) shows that we can only introduce a clock with the same time as an extant one. So s must have a clock reading t_1 and one reading t_2. But the subprocesses with both of these clocks are each capable of transitions, namely μ_1 and μ_2 respectively, so $\mathbf{clocks}(s) > 1$.

In the other direction, suppose that $\mathbf{clocks}(s) > 1$ for some s in $[\![P]\!]$. Let $P\,0 \to^\star s$ be the shortest sequence of transitions leading to a configuration s with $\mathbf{clocks}(s) > 1$. A simple induction on the length of this derivation shows that all clocks in s must read the same value t. As $\mathbf{clocks}(s) > 1$, a structural induction on s now shows that $s \xrightarrow[\delta_1]{\mu_1 @ t} s_1{}' \xrightarrow[\delta_2]{\mu_2 @ t} s_1$, for some actions μ_1, μ_2 and configurations $s_1{}'$ and s_1. This is the required ill–timed path. ☐

3.5. Independency in *ATTS*s

The concept of independency has been proposed by several authors, notably Bednarczyk [8], and Mazurkiewicz [36] as capturing concurrency information in a natural way. The idea is to give a relation ι over events, interpreting $\alpha\ \iota\ \beta$ as 'α and β are independent', i.e. could take place in parallel. We have a similar notion closely related to ill-timedness; suppose in an *ATTS*

$$s \xrightarrow[\delta_1]{\mu_1 @ t_1} s_1 \xrightarrow[\delta_2]{\mu_2 @ t_2} u \quad \text{and} \quad t_2 < t_1 + \delta_1$$

then the occurrence of μ_1 in the transition $s \xrightarrow[\delta_1]{\mu_1 @ t_1} s_1$ must be independent from that of μ_2 in $s_1 \xrightarrow[\delta_2]{\mu_2 @ t_2} u$ since it began before the other terminated. Thus in particular the path

$$s \xrightarrow[\delta_2]{\mu_2 @ t_2} u_1 \xrightarrow[\delta_1]{\mu_1 @ t_1} u$$

should also be possible, for some u_1; this is called the diamond property by Bednarczyk, and motivates the following.

Definition 17. Suppose that $G = (S, E, \to, s_0)$ is an *ATTS*. Then we say that G is *well–caused* if all ill–timed paths are due to concurrency, i.e. iff we can commute all independent actions

$$s \xrightarrow[\delta_1]{\mu_1 @ t_1} s_1 \xrightarrow[\delta_2]{\mu_2 @ t_2} u \quad \text{and} \quad t_2 \neq t_1 + \delta_1 \qquad \text{implies} \qquad \exists s_1' \in S\,.\, s \xrightarrow[\delta_2]{\mu_2 @ t_2} s_1' \xrightarrow[\delta_1]{\mu_1 @ t_1} u$$

For a *cIPA* process all time happens 'during' a transition, so there are no gaps in time; this is just urgency (and the reason we have $\neq$ rather than $<$ above). Hence we define

Definition 18. Suppose that $G = (S, E, \to, s_0)$ is an *ATTS*. Then we say that G is *timeful* iff the following conditions are all met.

(i) Its starting transitions all begin at time 0: $s_0 \xrightarrow[\delta]{\mu @ t} u \implies t = 0$.

(ii) While the process is running, something (perhaps a τ) is happening all the time: $s_0 \to \cdots \xrightarrow[\delta']{\mu' @ t'} s$ implies that for $0 \leq t \leq (t' + \delta')$ there exists a μ such that

$$s_0 \to \cdots \xrightarrow[\delta]{\mu @ t''} s' \to \cdots s \qquad \text{and} \qquad t \in [t'', t'' + \delta]$$

We can now show which class of *ATTS*s are the meanings of *cIPA* processes:

Proposition 19. If P is a *cIPA* process, then $\llbracket P \rrbracket$ is a timeful, well–caused, finite *ATTS*.

Proof. Timefulness can be easily shown by structural induction on P. In order to prove that the well–caused property holds, it is sufficient to show that for all $s \in \mathcal{C}(\textit{cIPA})$,

$$s \xrightarrow[\delta_1]{\mu_1 @ t_1} s_1 \xrightarrow[\delta_2]{\mu_2 @ t_2} u \quad \text{and} \quad t_2 \neq t_1 + \delta_1 \qquad \text{implies} \qquad \exists s_1' \in S \,.\, s \xrightarrow[\delta_2]{\mu_2 @ t_2} s_1' \xrightarrow[\delta_1]{\mu_1 @ t_1} u$$

Again, a structural induction on s suffices. □

4. Action Refinement

Action refinement,—the operation of replacing an action by a process,—has recently been the object of much interest in concurrency theory. Here we show that our durationful actions allow a particularly simple definition of action refinement. The technical development we present is inspired by [24] and [20, §3.6].

We shall, following [26], think of action refinement as a tool for structuring the meanings of processes; a high-level description of a complete process can be given, then further detail can be exposed by action refinement. Thus our notion of action refinement will be a *semantic* one.

Definition 20. A process P is a *valid refinement* of an action a, written P **refines** a, iff (every execution of) the process lasts the same time as the action: $\mathbf{dur}(P) = \{\Delta(a)\}$.

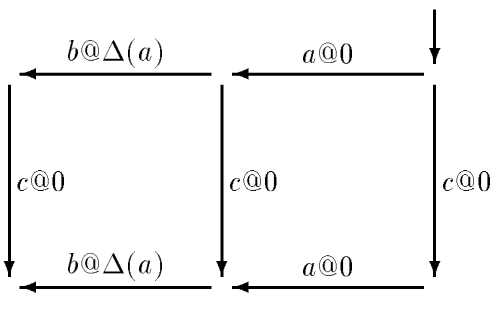
This is rather a strict notion of refinement; we even forbid processes that are always quicker than an action from being valid refinements of it. We do this partly for technical reasons, and partly to emphasise that speed–up is not always desirable; there are protocols which work at some speeds and fail at faster rates [4]. Note that we do not allow for refinement of actions by *cIPA* processes which are rooted branching bisimilar to *Nil* (since $\Delta(a) > 0$ for all a, and such processes have $\{0\}$ for their **dur**). Moreover, we do not consider refinements for processes including restriction (as here **dur** is not defined); this is eminently reasonable, as restriction is non-compositional with respect to timing. Finally, notice that, following [20, 24], internal actions are not refined.

Definition 21 (Semantic Substitution). Let $F : A \to \textit{cIPA}$ be a mapping from actions to *cIPA* processes with the property that

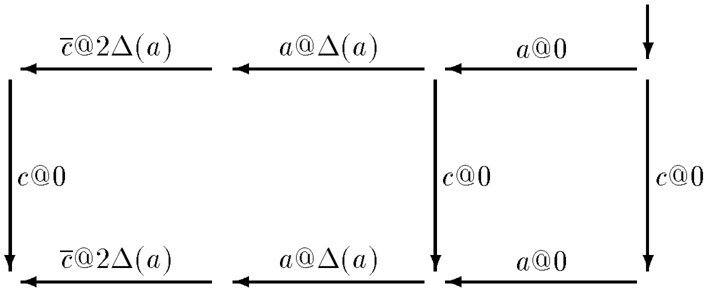
$$F(a) \;=\; P \qquad \text{implies} \qquad P \ \mathbf{refines}\ a$$

Then, we call F a *semantic substitution*, and for a *ATTS* G we define the refined graph $F(G)$ as follows: for every edge $s \xrightarrow[\delta]{a @ t} s'$ in G, take a copy $F(a)_i$ of $\llbracket F(a) \nearrow^{t} \rrbracket$. Identify s with the root node of $F(a)_i$ and s' with the end nodes of $F(a)_i$, and remove the edge $s \xrightarrow[\delta]{a @ t} s'$.

Example. Consider the process $a \,.\, b \;||\; c$. The meaning of this process as a process graph is depicted below:



Now suppose that $\Delta(b) = \Delta(a) + \Delta(c)$ and consider the mapping $F : A \to$ cIPA which maps b to $a\,.\,\overline{c}$, and acts like the identity on all the other actions. This is clearly a well–defined semantic substitution: the result of applying it to the process graph above is shown below.



Note that this process graph is indeed the result of the semantic substitution of a graph, namely $[\![a\,.\,\overline{c} \nearrow^{\Delta(a)}]\!]$, for the arcs corresponding to occurrences of action b in the original process graph.

The reader will have noticed that the process graph obtained by applying a semantic substitution F to a timeful, well-caused *ATTS* is timeful, but not necessarily well-caused. (Indeed, the above process graph provides an example of this phenomenon.) Hence, well-caused high-level descriptions of processes may turn out not to be well-caused when further details about their computations are unveiled by means of semantic substitutions. This is only to be expected because the definition of semantic substitution we use preserves the *non-interference* property of atomic actions, i.e. actions have no intermediate state for other activities [11, 13].

Our aim is to show that RBB is a congruence of semantic substitution. To do this, we need to see how to define a rooted branching bisimulation between refined process graphs.

Definition 22. Suppose that G, H are *ATTS*s and that R is a RBB witnessing $G \approx H$. Suppose further that we refine both G and H using some semantic substitution F. We define the refined RBB $F(R)$ as the smallest relation satisfying the conditions

(i) $R \subseteq F(R)$.

(ii) If $s \xrightarrow[\delta]{a@t} s'$ and $u \xrightarrow[\delta]{a@t} u'$ are edges in G and H respectively, s.t. $s\ R\ u$ and $s'\ R\ u'$, and both edges are replaced by copies $F(a)_1$ and $F(a)_2$ of $[\![F(a)\nearrow^t]\!]$ respectively, then nodes from $F(a)_1$ and $F(a)_2$ are related by $F(R)$ iff they are copies of the same node in $[\![F(a)\nearrow^t]\!]$.

Theorem 23. If $R : G \approx H$ is a RBB between process graphs G and H, and F is a semantic substitution, then $F(R)$ is a branching bisimulation between $F(G)$ and $F(H)$.

Proof. We check that $F(R)$ is indeed a RBB by an analysis of the cases in Definition 4:

(i) The root nodes of $F(G)$ and $F(H)$ are related by $F(R)$. It is easy to see that $F(R)$ only relates root nodes to root nodes, as this holds of R.

(ii) Assume that $F(R)$ relates s to u, and there is an edge $s \xrightarrow[\delta]{\mu@t} s'$ in $F(G)$. Then there are two cases:

  (1.) The nodes s and u originate from G and H. Then $s\ R\ u$, and we find either $\mu = \tau$ and $s \xrightarrow[\delta]{\tau@t} s'$ was already an edge in G, or G has an edge $s \xrightarrow[\delta']{a@t} s''$ and $s \xrightarrow[\delta]{\mu@t} s'$ is a copy of an initial edge from $[\![F(a)\nearrow^t]\!]$.
  In the first case, either $s'\ R\ u$ and hence $s'\ F(R)\ u$ or there is a path in H $u \Rightarrow u_1 \xrightarrow[\delta'']{\tau@t''} u'$ s.t. $s\ R\ u_1$ and $s'\ R\ u'$. This path also exists in $F(H)$ by definition, so $s\ F(R)\ u_1$ and $s'\ F(R)\ u'$ as required.

In the second case there must be a corresponding path $u \Rightarrow u_1 \xrightarrow[\delta']{a@t} u''$ in H s.t. $s \; R \; u_1$ and $s'' \; R \; u''$. Then, in $F(H)$ we find a path $u \Rightarrow u_1 \xrightarrow[\delta]{\mu@t} u'$ s.t. $s \; F(R) \; u_1$ and $s' \; F(R) \; u'$.

(2.) The nodes s and u originate from related copies $F(a)_i$ and $F(a)_j$ of some substituted graph $[\![F(a) \nearrow^{t'}]\!]$. Then $s \xrightarrow[\delta]{\mu@t} s'$ is an edge in $F(a)_i$ and s and u are copies of the same node in $[\![F(a) \nearrow^{t'}]\!]$. So, there is an edge $u \xrightarrow[\delta]{\mu@t} u'$ in $F(a)_j$ where u' is a copy of the same node in $[\![F(a) \nearrow^{t'}]\!]$ that s' is a copy of, and $s' \; F(R) \; u'$.

(iii) The case of an edge in $R(H)$ follows symmetrically.

□

5. An Algebraic Characterization of $\approx$

The purpose of this section is to axiomatize the congruence relation $\approx$ defined in the previous section over the language cIPA. Given the fundamental rôle played by configurations in defining the semantics of cIPA processes, we shall provide a complete axiomatization of $\approx$ over the set of configurations $\mathcal{C}$(cIPA). The key to the axiomatization presented in this section is the realization that the interpretation of processes given by an action-timed transition system is just an ordinary labeled transition system over a set of actions. The only difference being that the actions are *structured*, as they carry information on the timing of their occurrence and their duration.

Following van Glabbeek and Weijland [20, 21], there is a standard way of axiomatizing rooted branching bisimulation-like relations over ordinary, finite, acyclic labeled transition systems. The application of their method to cIPA involves the reduction of terms to (some syntactic notation for) *trees* over the set of actions into consideration. However, in our *ATTS* semantics processes evolve by performing events in E and these are not in the signature for configurations, so this method is not directly applicable to the language $\mathcal{C}$(cIPA). In order to apply van Glabbeek and Weijland's algebraic characterization to provide an axiomatization for $\approx$ over $\mathcal{C}$(cIPA), we thus need to extend the language $\mathcal{C}$(cIPA) to $\mathcal{EC}$(cIPA), where $\mathcal{EC}$(cIPA) is built as $\mathcal{C}$(cIPA) with the additional formation rule:

$$\alpha \in E \text{ and } s \in \mathcal{EC}(\text{cIPA}) \quad \Longrightarrow \quad \alpha : s \in \mathcal{EC}(\text{cIPA})$$

Thus the signature of the language $\mathcal{C}$(cIPA) has been extended by allowing prefixing operators of the form $\alpha : \_$, for $\alpha \in E$. The language $\mathcal{EC}$(cIPA) thus allows one to prefix timed, durationful events to configurations and this is what will be needed to define a suitable notation for trees.

The extended set of configurations $\mathcal{EC}$(cIPA) inherits the structural congruence $\equiv$ from its sublanguage $\mathcal{C}$(cIPA), and in what follows $\mathcal{EC}$(cIPA) will always be considered modulo $\equiv$. We shall use $s, u, w, s', \ldots$ to range over $\mathcal{EC}$(cIPA) and α, β to range over E. The operational semantics for $\mathcal{EC}$(cIPA) is obtained by extending the rules in displays 1 and 2 with the axiom

$$\textsc{Pre} \quad \frac{}{(\mu, t, \delta) : s \xrightarrow[\delta]{\mu@t} s}$$

It is easy to see that $\approx$ can be conservatively extended to the language $\mathcal{EC}$(cIPA) and that the following proposition holds:

Proposition 24. $\approx$ is a congruence over $\mathcal{EC}$(cIPA).

Proof. A straightforward extension to $\mathcal{EC}$(cIPA) of the proof of Proposition 8. □

Definition 25 (Sumforms). The set of *sumforms* over E is the least subset of $\mathcal{EC}$(cIPA) such that the following hold:

(i) *NIL* is a sumform (recall that we identify *NIL* with *NIL* 0);

$$
\begin{array}{rcll}
s + u & = & u + s & (A1) \\
(s + u) + w & = & s + (u + w) & (A2) \\
s + s & = & s & (A3) \\
s + \textit{Nil} & = & s & (A4) \\[1ex]
\alpha : (\tau : (s + u) + s) & = & \alpha : (s + u) & (H) \\[1ex]
(P + Q)\, t & = & P\, t + Q\, t & (S1) \\
(P \backslash C)\, t & = & (P\, t)\backslash C & (S2) \\
(P \mid\mid Q)\, t & = & (P\, t) \mid\mid (Q\, t) & (S3) \\[1ex]
(a \,.\, P)\, t & = & (a, t, \Delta(a)) : (P\, (t + \Delta(a))) & (R1) \\
(\textit{Wait}\; t' \,.\, P)\, t & = & \tau : (P\, (t + t')) & (R2) \\
\textit{Nil}\, t & = & \textit{Nil} & (R3) \\
(s + u)\backslash C & = & s\backslash C + u\backslash C & (R4) \\
((\mu, t, \delta) : s)\backslash C & = & \textit{Nil} \qquad\qquad \text{if } \mu \in C \cup \overline{C} & (R5) \\
((\mu, t, \delta) : s)\backslash C & = & (\mu, t, \delta) : s\backslash C \qquad \text{if } \mu \notin C \cup \overline{C} & (R6)
\end{array}
$$

$$
\left(\sum_{i \in I} (\mu_i, t_i, \delta_i) : s_i\right) \mid\mid \left(\sum_{j \in J} (\mu'_j, t'_j, \delta'_j) : u_j\right) = \qquad (\textrm{Int})
$$

$$
\sum_{i \in I} (\mu_i, t_i, \delta_i) : \left(s_i \mid\mid \left(\sum_{j \in J} (\mu'_j, t'_j, \delta'_j) : u_j\right)\right)
$$

$$
+ \sum_{j \in J} (\mu'_j, t'_j, \delta'_j) : \left(\left(\sum_{i \in I} (\mu_i, t_i, \delta_i) : s_i\right) \mid\mid u_j\right) + \sum_{(i,j):\, \mu_i = \overline{\mu_j}, t_i = t_j} \tau : (s_i \mid\mid u_j)
$$

Display 3. Equations over configurations.

(ii) if $\alpha \in E$ and s is a sumform then $\alpha : s$ is a sumform;

(iii) if s and u are sumforms, so is $s + u$.

In order to give a complete axiomatization for Rbb over $\mathcal{EC}(\textit{cIpa})$ (and, consequently, over *cIpa*), it will be sufficient to devise a set of axioms which allow us to reduce terms in $\mathcal{EC}(\textit{cIpa})$ to sumforms and which are complete for $\approx$ over sumforms, i.e. over finite E-labeled trees. Formally, the theory we shall consider is the two-sorted theory consisting of the set of axioms Eq over $\mathcal{EC}(\textit{cIpa})$ given in display 3 together with the following Tau axiom over E

Tau $\qquad (\tau, t, \delta) = (\tau, t', \delta')$

In view of the above axiom, we shall abbreviate all timed, durationful τ-actions to τ. (So that the axiomatisation we give is two-sorted.) The equivalence relation over E generated by Tau axiom will be denoted by $\equiv_E$.

The interplay between the equational theory of actions and that for processes is stated by the following substitutivity rule:

Sub $\qquad \alpha \equiv_E \beta$ and $s = u$ implies $\alpha : s = \beta : u$

We shall write, for s, u extended configurations, $s =_{\mathcal{C}} u$ iff the equality $s = u$ can be derived using equational logic from the equations in display 3 and the rule Sub.

A few comments on the axioms in display 3 are now in order. Axioms (A1)–(A4) and (H) are all that is needed to completely axiomatize rooted branching bisimulation over E-labeled finite trees [20, 21]. Axioms (S1)–(S3) are the structural axioms over the set of generators of

$\mathcal{C}(c\textsc{Ipa})$ and $\mathcal{EC}(c\textsc{Ipa})$. These axioms, together with (R1)–(R6) and (INT), are used to reduce configurations to finite E-labeled trees.

We can now state the promised completeness theorem:

Theorem 26. For all $s, u \in \mathcal{EC}(c\textsc{Ipa})$, $s \approx u$ iff $s =_{\mathcal{C}} u$. In particular, for all $c\textsc{Ipa}$ processes P, Q, $P \approx Q$ iff $P\,0 =_{\mathcal{C}} Q\,0$.

Proof. (Outline.) The proof of this result can be given following standard lines. Indeed the result follows from the following statements:

Soundness: For all $s, u \in \mathcal{EC}(c\textsc{Ipa})$, $s =_{\mathcal{C}} u$ implies $s \approx u$;

Completeness for sumforms [20, 21]: For all sumforms s, u, $s \approx u$ implies $s =_{\mathcal{C}} u$ can be proved using axioms (A1)-(A4) and (H); and

Normalization: Every $s \in \mathcal{EC}(c\textsc{Ipa})$ is provably equal to a sumform, i.e. for each $s \in \mathcal{EC}(c\textsc{Ipa})$ there exists a sumform s' such that $s =_{\mathcal{C}} s'$.

The soundness of the equations in display 3 can be easily shown by exhibiting appropriate rooted branching bisimulations. Indeed, all the equations but (H) are sound with respect to strong bisimulation equivalence [37].

The completeness of axioms (A1)-(A4) and (H) for branching bisimulation over finite trees has been proven by van Glabbeek and Weijland in [20, 21].

Finally, the normalization result can be proved in standard fashion by applying equations (S1)–(S3), (R1)–(R6) and (INT) as rewrite rules from left to right. □

Other authors, notably Ferrari et al. [18], have noted that expansion theorems often hold in the noninterleaving setting when the algebraic structure of transitions is taken into account; here we have shown that timing and duration are sufficient provided that we express the relationship between configurations rather than processes. Our account (INT) is a straightforward adaptation of Milner's interleaving law to our setting. Indeed, we could treat a different notion of synchronisation merely by presenting the appropriate operational rule and modifying (INT); this would, for instance, allow us to axiomatise the loose notion of timed synchronisation latterly proposed in [19, 27]. It should be noted, however [*op cit.*], that neither RBB nor indeed the notion of equivalence those author present are congruences of parallel composition, so such changes to our definitions are of doubtful merit.

6. *ATTS*s and Other Models of Concurrency

In this section we first give a broad overview of the relationship between the work reported here and salient other models in the literature. We then, to make a precise connection, relate our equivalence $\approx$ for a class of *concrete* processes to several other noninterleaving equivalences proposed in the literature. To reinforce our claim that timing information captures independency, we show how a class of well–behaved *ATTS*s can be translated into the asynchronous transition systems of Bednarczyk.

It is possible to classify noninterleaving behavioural theories for process algebras by means of the information they use to distinguish parallel processes from purely sequential ones. To begin with, we indicate some of the main notions in the process–algebraic literature and their relationship to our own, making no claim of completeness. As an aid to this discussion, we shall make use of the standard example in the literature, namely the processes $P = a \parallel b$ and $Q = a\,.\,b + b\,.\,a$.

- Boudol et al. [12] argue for the use of distribution information to distinguish parallelism from sequential nondeterminism. The processes P and Q are distinguished in this approach because there are two *locations* in P and only one in Q.

- Hennessy and the first author [2, 29] use abstract *duration* information to distinguish parallel processes from sequential ones. In this approach, P is distinguished from Q since it can start b before a has ended whereas Q cannot.
- Darondeau and Degano [15] use information about the *causal* structure of processes to distinguish P (which has no causal structure) from Q (where a causes b or b causes a).

Our position is rather complex in this classification; using timing and duration enables us to *discover* which states are independent. Thus our *ATTS*s, like various 'true concurrency' versions of transition systems [8, 46], incorporate independency information, as we have shown. To generate these transition systems, we have used the notion of local time, which is clearly related to that of location.

Notice, incidentally, that both the location and causal approaches use annotated tree structures, as do we. We are clearly seeing the renaissance of tree–based approaches to concurrency semantics, as workers discover how to abandon the strictures of pure interleaving without leaving the convenience of trees. Indeed, this is the essence of our complete axiomatisation: we lift an untimed result to our setting. Many more such liftings are possible—for instance, we could use the efficient algorithm for deciding branching bisimulation of [28] in our setting.

6.1. *RBB* over concrete processes

The class of concrete processes (i.e. processes without internal transitions [5]) built from the operators *NIL*, action-prefixing, choice and parallelism (without synchronization) is a particularly well–behaved one; it is already known that causal bisimulation equivalence [15], t-observational equivalence [2, 29], ST-bisimulation equivalence [20] and location equivalence [12] coincide for these processes. Here we show that these equivalences also coincide with $\approx$. (Note that, over concrete processes, $\approx$ is just standard strong bisimulation equivalence over the associated *ATTS*s.)

In view of the results in [1], it is sufficient to show that $\approx$ shares the same finite axiomatization of the above-mentioned congruences over concrete processes. To carry out this programme, it is necessary to extend the language for concrete processes with Bergstra and Klop's left-merge operator [9]. The syntax of concrete *cIPA* is then defined by

$$P \ ::= \ a\,.\,P \mid P + P \mid \textit{NIL} \mid P \,||\, P \mid P \,\lfloor\!\lfloor\, P$$

and its associated set of concrete configurations is defined as follows:

$$s \ ::= \ P\,t \mid s + s \mid s \,||\, s \mid s \,\lfloor\!\lfloor\, s$$

The reader will have no trouble in convincing himself/herself that a suitable notion of canonical term can be easily defined over concrete configurations by assuming that the operators $\_\ t$ distribute over the left-merge operator, as well as choice and parallel composition. (See section 2.5 for details.)

The operational semantics of concrete *cIPA* is obtained by adding the following rule to rules ACT, CL, CR, PAL and PAR in displays 1 and 2:

$$\textsc{Left}\ \frac{s_1 \xrightarrow[\delta]{a@t} s_1'}{s_1 \,\lfloor\!\lfloor\, s_2 \xrightarrow[\delta]{a@t} s_1' \,||\, s_2}$$

We will now show that the axioms that completely axiomatize causal bisimulation equivalence, location equivalence, t-observational equivalence and ST-bisimulation equivalence over concrete *cIPA* (see [1, Figure 3]) are also sound and complete for $\approx$. For ease of reference, the set of equations we will consider is listed in display 4.

Let $\doteq$ denote the least congruence over concrete *cIPA* processes which satisfies the equations in display 4.

Proposition 27. For all concrete *cIPA* processes P, Q, $P \doteq Q$ implies $P \approx Q$.

$$
\begin{array}{rcll}
P+Q & = & Q+P & (A1)\\
(P+Q)+R & = & P+(Q+R) & (A2)\\
P+P & = & P & (A3)\\
P+\textsc{Nil} & = & P & (A4)\\[1ex]
(P+Q) \mathbin{|\!\!\lfloor} R & = & P \mathbin{|\!\!\lfloor} R + Q \mathbin{|\!\!\lfloor} R & (LM1)\\
(P \mathbin{|\!\!\lfloor} Q) \mathbin{|\!\!\lfloor} R & = & P \mathbin{|\!\!\lfloor} (Q \,||\, R) & (LM2)\\
P \mathbin{|\!\!\lfloor} \textsc{Nil} & = & P & (LM3)\\
\textsc{Nil} \mathbin{|\!\!\lfloor} P & = & \textsc{Nil} & (LM3)\\[1ex]
P \,||\, Q & = & P \mathbin{|\!\!\lfloor} Q + Q \mathbin{|\!\!\lfloor} P & (PAR)
\end{array}
$$

Display 4. Equations over concrete processes.

In the proof of completeness of $\doteq$ with respect to $\approx$ over concrete processes, we will follow the lines of similar results in [1, 2, 14], and will make a fundamental use of the following key decomposition property: for $t > 0$

Dec $\qquad P\,t \;||\; P'\,0 \;\approx\; Q\,t \;||\; Q'\,0 \quad \text{implies} \quad P \approx Q \quad \text{and} \quad P' \approx Q'$

An elegant proof of statement Dec can be given by adapting the unique factorisation result presented in [39, Theorem 4.2.2] to our setting. This we now present, referring the reader to [2, 38, 39] for more details and intuition on unique factorization results.

Definition 28. Let s be a concrete configuration. Then s is said to be:

- *irreducible* if $s \approx s_1 \,||\, s_2$ implies $s_1 \approx \textsc{Nil}$ or $s_2 \approx \textsc{Nil}$;
- *prime* if s is irreducible and $s \not\approx \textsc{Nil}$.

For each concrete configuration s, a parallel factorization for s modulo $\approx$ is given by a set $\{s_1, \ldots, s_k\}$, $k \geq 0$, of primes such that

$$s \;\approx\; s_1 \,||\, \cdots \,||\, s_k$$

where, by convention, $s_1 \,||\, \cdots \,||\, s_k$ is $\textsc{Nil}$ if $k = 0$.

Proposition 29. Each concrete configuration s may be expressed, modulo $\approx$, as a unique parallel composition of primes.

Proof. A straightforward adaptation of the proof of Theorem 4.2.2 in [39]. The proof makes an essential use of the following version of the so-called *simplification lemma*, a result first proved in [14] for distributed bisimulation equivalence and subsequently adapted to strong bisimulation in [39, Lemma 4.2.1]:

$$s_1 \,||\, s \;\approx\; s_2 \,||\, s \qquad \text{implies} \qquad s_1 \;\approx\; s_2$$

□

We can now prove the promised decomposition result Dec.

Proposition 30. For all concrete cIPA processes P, P', Q, Q' and $t \in \mathbb{R}^+$,

$$P\,t \;||\; P'\,0 \;\approx\; Q\,t \;||\; Q'\,0 \qquad \text{implies} \qquad P \approx Q \quad \text{and} \quad P' \approx Q'$$

Proof. The claim is easily seen to hold if any of the processes P, P', Q, Q' is equivalent to $\textsc{Nil}$.

Assume then that P, P', Q, Q' are not equivalent to $\textsc{Nil}$. By Proposition 29, there exist unique parallel factorizations for $P\,t$, $P'\,0$, $Q\,t$ and $Q'\,0$. Let us assume that these are given by

$$
\begin{array}{rcccc}
P\,t & \approx & s_1^t & ||\cdots|| & s_k^t\\
P'\,0 & \approx & s_1 & ||\cdots|| & s_h\\
Q\,t & \approx & w_1^t & ||\cdots|| & w_m^t\\
Q'\,0 & \approx & w_1 & ||\cdots|| & w_n
\end{array}
$$

By the proviso of the proposition, substitutivity and Proposition 29, we have that, modulo $\approx$,

$$\{s_1^t, \ldots, s_k^t, s_1, \ldots, s_h\} \;=\; \{w_1^t, \ldots, w_m^t, w_1, \ldots, w_n\}$$

It is now easy to see that none of the prime factors for $P\,t$, s_i^t, is equivalent to any of the w_j's. In fact, any initial transition from s_i^t will start at time $t > 0$, whilst all the transitions from w_j will start at time 0. Moreover, as $P \not\approx$ NIL, P as at least one transition, and so does each s_i^t. Similarly, none of the prime factors for $Q\,t$ is equivalent to any of the s_j's. This implies that $\{s_1^t, \ldots, s_k^t\}$ and $\{w_1^t, \ldots, w_m^t\}$ are identical parallel factorizations, and so are $\{s_1, \ldots, s_h\}$ and $\{w_1, \ldots, w_n\}$. By substitutivity, we then have that $P\,t \approx Q\,t$ and $P'\,0 \approx Q'\,0$.

To complete the proof, we are thus left to show that $P\,t \approx Q\,t$ implies $P\,0 \approx Q\,0$. However, this is easily seen to hold using Lemma 6. □

We now have enough technical machinery to prove the completeness result for concrete cIPA. The proof of this result relies, as usual, on the isolation of suitable normal forms for processes. These take the form

$$\sum_{i \in I} a_i \,.\, P_i \;\lfloor\!\lfloor\; P_i'$$

where I is a finite index set, and P_i, P_i' are themselves normal forms.

Theorem 31. For all concrete cIPA processes P, Q, $P \doteq Q$ iff $P \approx Q$.

Proof. The 'only if' implication (soundness) is just Proposition 27. To prove the 'if' implication (completeness), we proceed by induction on the combined size of P and Q. By standard normalization results for concrete processes, such as Lemma 1.3.5 in [14], we can safely restrict ourselves to proving completeness for normal forms.

Assume then that $P = \sum_{i \in I} a_i \,.\, P_i \lfloor\!\lfloor P_i' \approx \sum_{j \in J} b_j \,.\, Q_j \lfloor\!\lfloor Q_j' = Q$. We will now show that $Q + P \doteq Q$. The claim will then follow by symmetry and transitivity.

To prove $Q + P \doteq Q$, it is sufficient to show that for each summand $a_i \,.\, P_i \lfloor\!\lfloor P_i'$ of P,

$$Q + a_i \,.\, P_i \;\lfloor\!\lfloor\; P_i' \doteq Q$$

To see that this is indeed the case, note that $P\,0 \xrightarrow[\Delta(a_i)]{a_i@0} P_i\,\Delta(a_i) \,||\, P_i'\,0$. As $P \approx Q$, there exists $j \in J$ such that $Q\,0 \xrightarrow[\Delta(b_j)]{b_j@0} Q_j\,\Delta(b_j) \,||\, Q_j'\,0$, $P_i\,\Delta(a_i) \,||\, P_i'\,0 \approx Q_j\,\Delta(b_j) \,||\, Q_j'\,0$, and $a_i = b_j$. By Proposition 30 and induction, we now have that $P_i \doteq Q_j$ and $P_i' \doteq Q_j'$. The claim now follows immediately by substitutivity and equations (A1)–(A4). □

As an immediate corollary of this result, we now have that:

Corollary 32. For concrete processes, $\approx$ coincides with causal bisimulation equivalence [15], ST-bisimulation equivalence [20], t-observational equivalence [2, 29] and location equivalence [12].

Proof. By the results in [1] and Theorem 31, $\approx$ shares the same finite axiomatization of the above-mentioned congruences over concrete processes. □

It is worth noting that, by the above result, $\approx$ does not depend on the choice of duration function over concrete processes, regardless of whether the duration function is constant or not. Proposition 13 thus holds in a sharpened version over concrete processes.

Further comparison is slightly hindered by our decision to use *branching* bisimulation rather than ordinary strong bisimulation; this makes it harder to extract the features of our equivalence that are due to timing, and those that arise through our sensitive treatment of branching structure. However, we can state that for processes with internal transitions and an appropriate choice of duration function, $\approx$ distinguishes more purely sequential processes than strong bisimulation equivalence (see the restriction example in section 3.1) and, a *fortiori*, than all of the noninterleaving equivalences we are aware of. Moreover, it is incomparable with location equivalence. In fact, regardless of the duration of actions a and b, it would identify the processes

$$P = (a\,.\,\sigma\,.\,c \,||\, b\,.\,\overline{\sigma}\,.\,d)\backslash\{\sigma\} \qquad \text{and} \qquad Q = (a\,.\,\sigma\,.\,d \,||\, b\,.\,\overline{\sigma}\,.\,c)\backslash\{\sigma\}$$

which are distinguished by location equivalence. Thus we have:

Proposition 33. For full CCS, $\approx$ is, in general, incomparable with the weak versions of causal bisimulation equivalence, location equivalence and ST-bisimulation equivalence.

Proof. Consider $P = (a\,.\,b \,||\, \overline{a})\backslash\{a\}$ and $Q = (c\,.\,b \,||\, \overline{c})\backslash\{c\}$. Then, if $\Delta(a) \neq \Delta(c)$, we have that $P \not\approx Q$. On the other hand, P and Q are related by causal bisimulation equivalence, location equivalence and ST-bisimulation equivalence.

Conversely, consider the processes $P = (a\,.\,b \,||\, \overline{b}\,.\,c)\backslash\{b\}$ and $Q = a\,.\,\text{\textsc{Nil}}$. Then $P \approx Q$, whilst P and Q are distinguished by causal bisimulation equivalence, location equivalence and ST-bisimulation equivalence. □

6.2. Translating *ATTS*s into ATSs

We now turn to the matter of the information that timing gives us, showing that it alone allows us to deduce the causal structure of transitions. We do this by presenting a relation $\bowtie$ defined over *ATTS*s which captures which transitions are independent, noting *en passant* that timing information, like the annotated transitions of [17], is also sufficient for an analysis of causality. We show in detail that $\bowtie$ does indeed allow us to define an asynchronous transition system from a suitably well–behaved *ATTS*.

To begin the account of how timing information captures the idea of independency, we recall the formal definition of the asynchronous transition systems (or ATSs) of [8]:

Definition 34. A finite rooted asynchronous transition system $\mathbf{Q}$ is a tuple $(Q, T, \Rightarrow, q_0, \iota)$ where

- $(Q, T, \Rightarrow, q_0)$ is a rooted transition system with Q and T finite sets, and root q_0;
- $\iota \subset T \times T$ is a symmetric, irreflexive relation satisfying the *diamond property* [8]:

$$s \xrightarrow{e} u \xrightarrow{f} s' \quad \text{and} \quad e\,\iota\,f \qquad \text{implies} \qquad \exists u' \in Q \quad \text{such that} \quad s \xrightarrow{f} u' \xrightarrow{e} s'$$

We will, as usual, confine attention to ATSs satisfying the following conditions:

unambiguity This property essentially says that the same transition cannot take us to different places:

$$s \xrightarrow{e} u \quad \text{and} \quad s \xrightarrow{e} u' \qquad \text{implies} \qquad u = u'$$

forward stability This says that if we have two independent actions possible next, then that is as a result of parallelism, so after taking one the other must still be possible: $s \xrightarrow{e} u$ and $s \xrightarrow{e'} u'$ and $e\,\iota\,e'$ together imply

$$\exists s' \text{ such that } \quad u \xrightarrow{e'} s' \quad \text{and} \quad u' \xrightarrow{e} s'$$

lack of cycles This is essentially the same property as was required for *ATTS*s in definition 1.

reachability Likewise.

We are now ready to translate an *ATTS* into an ATS. Suppose $\mathbf{S} = (S, E, \rightarrow, s_0)$ is a well-caused, timeful *ATTS*. The events $e \in E$ are non-unique in the sense that we can have $s \xrightarrow{e} s'$ and $u \xrightarrow{e} u'$ without $s = u$ and $s' = u'$. Moreover, even if $s = u$, we do not necessarily have $s' = u'$. Thus consider the structure $\mathbf{S}' = (S, E', \rightarrow', s_0)$ where

$$E' = \rightarrow\; \subset S \times E \times S$$

is the set of transitions in $\mathbf{S}$, and

$$s \xrightarrow{(u,e,u')}{}' \, s' \qquad \text{iff} \qquad s \xrightarrow{e} s' \quad \text{and} \quad u = s \quad \text{and} \quad u' = s'$$

Immediately we have that $\mathbf{S}'$ is a rooted transition system, which is deterministic, acyclic and satisfies the reachability property.

Our next task is to define a suitable notion of independence over this transition system. Clearly, if $e' = (s, e, s')$, $f' = (s', f, u')$ and there exists an ill-timed path $s \xrightarrow{e} s' \xrightarrow{f} u'$ in $\mathbf{S}$, then e' and f' should be independent. But this relation is not symmetric, for two reasons; firstly the commuted path (which we know exists as $\mathbf{S}$ is well-caused)

$$s \xrightarrow{(s,f,u)'} u \xrightarrow{(u,e,u')'} u'$$

may (fortuitously) not be ill–timed, and secondly, the 'events' (s, e, s') and (u, e, u') are different, while we want them to be the same in the generated ATS. Our solution is to quotient E' by a suitable congruence:

Definition 35. Define for $\mathbf{S}'$ the relation $\kappa \subset E' \times E'$ as $e' \; \kappa \; f'$ iff $e' = (s, e, s')$, $f' = (s', f, u')$ and there exists an ill-timed path $s \xrightarrow{e} s' \xrightarrow{f} u'$ in $\mathbf{S}$. Suppose $s \xrightarrow{(s,f,u)'} u \xrightarrow{(u,e,u')'} u'$ is the diamond guaranteed by well-causedness or vice versa. Extend κ and define $\lambda \subset E' \times E'$ by saying $(s, f, u) \; \kappa \; (u, e, u')$, $(s, f, u) \; \lambda \; (s', f, u')$ and $(s, e, s') \; \lambda \; (u, e, u')$ whenever this happens, and write λ^* for the reflexive and transitive closure of λ.

Proposition 36. The relation λ^* is an equivalence relation, and $\bowtie$ defined over λ^*–equivalence classes by $[e'] \bowtie [f']$ iff $\exists e'' \in [e'], f'' \in [f']$ such that $e'' \; \kappa \; f''$ is a symmetric and irreflexive relation.

Proof. That λ^* is an ER is trivial. To show that $\bowtie$ is an independence relation, first note that $e'' \; \kappa \; f''$ implies $\neg(e'' \; \lambda^* \; f'')$ and $e'' \; \lambda^* \; f''$ implies $\neg(e'' \; \kappa \; f'')$, as a simple geometrical argument reveals. Thus $\bowtie$ is irreflexive. Symmetry then follows from the definition of κ. □

Definition 37. The asynchronous transition system associated with $\mathbf{S}$, written $\mathcal{U}(\mathbf{S})$, is defined as $(S, E'/\lambda^*, \rightarrow', s_0, \bowtie)$, where $\rightarrow'$ is defined as $(s, e', s') \in \rightarrow'$ if and only if $(s, e'', s') \in \rightarrow$ and $e' = [e'']/\lambda^*$.

Proposition 38. If $\mathbf{S}$ is an *Atts*, then $\mathcal{U}(\mathbf{S})$ is an ATS.

Proof. We have to verify:

(i) That $\bowtie$ is a symmetric, irreflexive relation; this is just proposition 36.
(ii) The diamond property. This follows as the generating *Atts* is well-caused.
(iii) Unambiguity. This follows by construction of E'.
(iv) Forward stability. By the definition of $\bowtie$, if two actions are independent, then a diamond exists between them.
(v) Lack of cycles and reachability. Again, clearly true by construction.

□

7. An Example

As an extended example of the use of *cIpa*, we present the specification of a remote data-gathering unit in the calculus. This arose through a collaboration of the second author with Patrick Peglar of Delta T Devices Ltd., and thus is a (simplification of a) real industrial example rather than one contrived just for presentation purposes.

7.1. The Problem

The purpose of a data-gatherer is to take measurements at prespecified times. In usual applications, a number of sensors are attached to a data-gatherer. Each sensor must be turned on at some given time before the measurement it takes can be made (typically so that thermocouples have time to stabilise). The sensor then takes a measurement and hands the data back to the data-gatherer. Sensors are often cheap and unsophisticated devices, without internal clocks or buffering, so the data-gatherer must ensure that the sensor is turned on at the right time, and that it is ready to receive the data when it is ready. Thus a sensor can be specified as

$$\textsc{Sensor}1 \stackrel{\mathrm{def}}{=} \textsc{Wait}\ t_1 \,.\, \overline{\textsc{Start}1} \,.\, \textsc{Wait}\ t_1' \,.\, \overline{\textsc{Read}1} \,.\, \textsc{Write}1$$

which is to say that it is a device that waits some unspecified time t_1 until it is turned on by a $\textsc{Start}1$ action. After being turned on by this action, it waits time t_1' before taking a measurement, which it then $\textsc{Write}$s.

The data-gatherer, then, will be responsible for doing a $\textsc{Start}1$ with the correct t_1', which will ensure that the sensor does a $\textsc{Read}1$ when required. It must also be ready $t_1' + \Delta(\textsc{Read}1)$ units of time after finishing the $\textsc{Start}1$ to do a $\overline{\textsc{Write}1}$, allowing the sensor to transfer data back to it. Finally, it $\textsc{Process}$es the data.

The design of the data-gatherer is only non-trivial when more than one sensor is considered, so suppose we also have

$$\textsc{Sensor}2 \stackrel{\mathrm{def}}{=} \textsc{Wait}\ t_2 \,.\, \overline{\textsc{Start}2} \,.\, \textsc{Wait}\ t_2' \,.\, \overline{\textsc{Read}2} \,.\, \textsc{Write}2$$

(In a real application, the requirement is recurrent; we have to make a series of measurements at each sensor, rather than just one. However, our design will extend smoothly to this setting, so we avoid the use of recursion to keep the complexity of the design down.)

7.2. Timing Analysis

Suppose that the two measurements $\textsc{Read}1$ and $\textsc{Read}2$ have deadlines d_1 and d_2 respectively, i.e. $\textsc{Read}i$ must happen at $t = d_i$. Clearly, then, we must have the deadline equation

$$d_i = t_i + \Delta(\textsc{Start}i) + t_i'$$

in order for the deadlines to be met.

Now, clearly the design of the data-gatherer will depend on the specific deadlines. For instance, $\textsc{Write}i$ starts at time $d_i + \Delta(\overline{\textsc{Read}i})$, so if the intervals

$$[d_1 + \Delta(\overline{\textsc{Read}1}), d_1 + \Delta(\overline{\textsc{Read}1}) + \Delta(\textsc{Write}1)]$$

and

$$[d_2 + \Delta(\overline{\textsc{Read}2}), d_2 + \Delta(\overline{\textsc{Read}2}) + \Delta(\textsc{Write}2)]$$

overlap, then a purely sequential implementation is not possible, as the data-gatherer will need to do both a $\overline{\textsc{Write}1}$ and a $\overline{\textsc{Write}2}$ simultaneously. Similarly, if we need to turn on both sensors simultaneously, we will end up needing a parallel implementation. Suppose, then, for the moment, that the intervals above do not overlap, and neither do

$$[t_1, t_1 + \Delta(\overline{\textsc{Start}1})] \qquad \text{and} \qquad [t_2, t_2 + \Delta(\overline{\textsc{Start}2})]$$

(which is eminently reasonable, as the $\textsc{Start}$ actions are usually of very short duration).

Then we can implement the data-gather in a sequential fashion (which is useful, as the resources for a parallel implementation are often not available). For convenience, suppose also that $d_1 < d_2$.

The data-gatherer waits until it needs to start the sensor with the smallest value of t_i, 2 say, switches that on, waits until it must start 1, switches that on, waits for the first bit of data, collects that, waits for the second, gets that, and then processes the data. A simple bit of arithmetic (and the use of **dur**) reveals that correct functioning depends on the behaviour shown in display 5.

Time	Sensor1 Action	Sensor2 Action	Data-Gatherer Action
0	Waiting	Waiting	Waiting
t_2	Waiting	$\overline{\textsc{Start}2}$	$\textsc{Start}2$
$t_2 + \Delta(\textsc{Start}2)$	Waiting[†]	Waiting	Waiting
t_1	$\overline{\textsc{Start}1}$	Waiting	$\textsc{Start}1$
$t_1 + \Delta(\textsc{Start}1)$	Waiting	Waiting	Waiting
d_1	$\textsc{Read}1$	Waiting	Waiting
$d_1 + \Delta(\textsc{Read}1)$	$\overline{\textsc{Write}1}$	?	$\textsc{Write}1$
d_2	?	$\textsc{Read}2$	?
$d_2 + \Delta(\textsc{Read}2)$	?	$\overline{\textsc{Write}2}$	$\textsc{Write}2$

Display 5. The timing constraints on the Data-Gatherer.

The ? occur because we do not know where warming up the second sensor occurs relative to taking the first measurement. This is a scheduling issue that depends on the relationship between the d_i and the times $\textsc{Start}$s happen. But the time between being activated and taking a measurement is a fixed (presumably physical) property of each sensor; call it $warm_i$. Immediately the urgency of $\textsc{Read}$s gives

$$warm_i = \Delta(\overline{\textsc{Start}i}) + t'_i$$

which fixes t'_i. Together with the deadline equation, it also fixes the order of the $\textsc{Start}$s and $\textsc{Read}$s, for

$$d_i = t_i + warm_i$$

which fixes t_i as d_i is given. Finally, suppose that the $warm_i$s are such that

$$0 < t_2 < t_1 < d_1 < d_2$$

This will allow us to give a sequential implementation to the data-gatherer.

7.3. Implementation

In this section we give the promised implementation:

$$\begin{aligned} \textsc{DataGatherer} \stackrel{\text{def}}{=} \; & \textsc{Wait}\ t_2 \,.\, \textsc{Start}2 \,.\, \textsc{Wait}\ r_1 \,.\, \textsc{Start}1 \,.\, \textsc{Wait}\ r_2 \,. \\ & \overline{\textsc{Write}1} \,.\, \textsc{Wait}\ r_3 \,.\, \overline{\textsc{Write}2} \,.\, \textsc{Process} \end{aligned}$$

where $r_1 = t_1 - t_2 - \Delta(\textsc{Start}2)$, $r_2 = t'_1 + \Delta(\textsc{Read}1)$ and $r_3 = (d_2 + \Delta(\textsc{Read}2)) - (d_1 + \Delta(\textsc{Read}1) + \Delta(\textsc{Write}1))$. In practice it would be helpful to make r_1 as short and r_2 as long as possible, by selecting sensors with appropriate $warm_i$s, so that we can shut (much of) the data-gatherer down, and save power, between the $\textsc{Start}$s and the $\textsc{Read}$s.

[†] Implicitly assuming that $t_2 + \Delta(\textsc{Start}2) < t_1$, which follows as the intervals $[t_1, t_1 + \Delta(\overline{\textsc{Start}1})]$ and $[t_2, t_2 + \Delta(\overline{\textsc{Start}2})]$ were assumed not to overlap in time. Note that $\textsc{Start}$ actions typically have very small durations.

Our description of this, sequential version of the system, is then

$$\textit{System} \overset{\text{def}}{=} (\textit{DataGatherer} \mid\mid \textit{Sensor}1 \mid\mid \textit{Sensor}2)\backslash\{\textit{Start}i, \textit{Write}i\}$$

Thus far, the example has demonstrated a feature of the *cIpa* synchronisation discipline; often we write a process including $\textit{Wait}\ t$ for some unspecified t, as we did with the $\textit{Sensor}i$s and t_is, and then fix the value of t so that some desired synchronisation can happen. In the above, we have also derived assumptions necessary on timing constraints to allow a sequential implementation. This practice corresponds well with informal design procedures in real–time systems, where delays are often inserted to allow some desired *rendez-vous*[‡] and timing properties exploited in an implementation.

7.4. Correctness

The correctness of the implementation (at least as far as it is captured by *cIpa*) can be demonstrated by showing that it is equivalent to a process which does a $\textit{Read}i$ at d_i for each i, and then, suitably later, *Process*es the data. However, our implementation is predicated on a set of timing assumptions that allow a sequential implementation, and these must be built into our specification.

The first read must begin at time d_1, and the second at d_2, which happens after d_1, so we have

$$\textit{Spec} \overset{\text{def}}{=} (\textit{Wait}\ d_1 \,.\, \textit{Read}1) \mid\mid (\textit{Wait}\ d_2 \,.\, \textit{Read}2 \,.\, \textit{Wait}\ r_4 \,.\, \textit{Process})$$

where $r_4 = \Delta(\textit{Write}2)$.

It is then routine, if rather tedious, to show that

Proposition 39. The implementation is equivalent to the specification; $\textit{Spec} \approx \textit{System}$.

8. Further Work

We conclude the paper with a discussion of some extensions to the work presented here.

8.1. Passive Actions

Urgency is a crucial property in *cIpa*; without it, many of our proofs would not be valid. However, it is quite constraining, especially given our choice of synchronisation rule. One way of allowing more freedom in the timing of actions without losing all hope of a tractable calculus is to add *lazy* actions; $\tilde{a}.P$, unlike $a \,.\, P$, waits to be triggered from a synchronisation [42]. Lazy actions like $\tilde{a}$ allow us to keep our synchronisation rule while allowing idling; $\tilde{a}.\textit{Nil}$ can synchronise with the $\overline{a}$ in $b \,.\, \overline{a} \,.\, \textit{Nil}$, whereas $a \,.\, \textit{Nil}$ cannot.

Lazy actions also add expressive power to the calculus via their interaction with choice; $\tilde{a} \,.\, \textit{Nil} + \tilde{b} \,.\, \textit{Nil}$ is rather like the CSP external choice between a and b, for instance.

It would be interesting to try to extend our results for *cIpa* to a calculus with both lazy and eager actions, some progress in this direction being reported in [42].

8.2. Recursion

In the main body of this paper, we have dealt with an algebra for finite processes. However, facilities for recursive definitions of processes are a vital ingredient of any process algebra. We can extend *cIpa* with recursive process definitions by allowing constants to be defined recursively by

---

[‡] Our use of $\textit{Wait}\ t$ with t undefined means that we are implicitly working with a design calculus with τ rather than $\textit{Wait}\ t$ prefixes which is compiled down to full *cIpa* syntax by fixing the duration of *Wait* s; the least syntactic overhead comes if we view this as we have above, rather than making the design calculus explicit.

means of equations $X \overset{\text{def}}{=} P$, where P is a process term built from *cIPA* operations and constants. The behaviour of these recursively defined processes is given by the following standard rule

$$\textsc{Rec}\ \frac{P\,t \xrightarrow[\delta]{\mu @ t} s}{X\,t \xrightarrow[\delta]{\mu @ t} s}\ X \overset{\text{def}}{=} P,\ X \text{ time–guarded in } P$$

where, loosely, an occurrence of a variable is time–guarded in a process expression if it is prefixed by an action or a wait; see [16] or [42] for a comprehensive account of unfolding time–guarded infinite behaviours.

Some of our results extend to recursive processes; compositionality (section 3.2) and time uniformity (section 3.3) are straightforward. Moreover, a simple structural induction shows that the meaning $[\![P]\!]$ of a recursive process P is a timeful and well–caused *ATTS*. It can then be shown that Zeno processes cannot be recursively defined; this follows since all processes are sort–finite[§] and time–guarded, and thus there is a bound on the number of simultaneous actions a given process can display at some time. However, our axiomatisation is inherently limited to finite processes, so further work, along the lines of [21], is necessary before a satisfactory equational account of $\approx$ can be given for recursive cIPA.

8.3. Other Operators

An industrial strength timed specification formalism would need a much larger set of operators than we have provided in *cIPA*. Most of these would be derived, but there is a crucial class of operations,—timeouts,—that cannot be coded in *cIPA*. The introductions of timeouts, such as those provided in timed CSP [16], would be a useful extension to *cIPA*; the correct intuition, however, is somewhat unclear: how should local clocks interact with timeouts? The right balance between generality and tractability is not clear to us here.

Acknowledgments

The second author would like to thank Gian Luigi Ferrari for a most stimulating visit to Pisa where this work was first presented. Both authors would like to thank Roberto Gorrieri, Matthew Hennessy and Vladimiro Sassone for helpful remarks.

The second author was partly funded by a Royal Society European Research Fellowship at the GMD Bonn, and partly by German Ministry for Science and Technology Verbund Projekt Korso (Grant no. ITS 900 IAT).

Bibliography

1. L. Aceto, *Relating distributed, temporal and causal observations of simple processes*, Fundamenta Informaticae, Volume 17 (1992), Number 4, Pp. 369–397.
2. L. Aceto and M. Hennessy, *Towards action refinement in process algebra*, Information and Computation, Volume 103 (1993), Number 2, Pp. 204–269.
3. L. Aceto and D. Murphy, *On the Ill–Timed but Well–Caused.* In CONCUR (E. Best, Ed.), Volume 715, Springer-Verlag LNCS, 1993.
4. T. Axford, *Concurrent Programming: Fundamental Techniques for Real-Time and Parallel Software Design*, Wiley, 1989.
5. J. Baeten and J. Bergstra, *Global renaming operators in concrete process algebra*, Information and Computation, Volume 78 (1988), Number 3, Pp. 205–245.
6. J. Baeten and J. Bergstra, *Real time process algebra*, Formal Aspects of Computing, Volume 3 (1991), Number 2, Pp. 142–188.

[§] A process P is sort–finite if the set of actions in *Act* labelling arcs in $[\![P]\!]$ is finite.

7. J. Baeten and W. Weijland, *Process algebra*, Cambridge Tracts in Theoretical Computer Science, Volume 18, Cambridge University Press, 1990.
8. M. Bednarczyk, *Categories of asynchronous systems*, Ph.D. thesis, Department of Computer Science, University of Sussex, 1987, available as Technical Report Number 3/87.
9. J. Bergstra and J. Klop, *Fixed point semantics in process algebras*, Report Number IW 206, Mathematisch Centrum, Amsterdam, 1982.
10. E. Best and M. Koutny, *Petri net semantics of priority systems*, Theoretical Computer Science, Volume 96 (1992), Pp. 175–215.
11. G. Boudol, *Atomic actions (note)*, Bulletin of the EATCS, Volume 38 (1989), Pp. 136–144.
12. G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn, *A theory of processes with localities*, Technical Report 13/91, Department of Computer Science, University of Sussex, 1991.
13. R. Campbell and P. Jalotte, *Atomic actions in concurrent systems*, in Proceedings of the 5th International Conference on Distributed Computing Systems, Pp. 184–191, 1985.
14. I. Castellani and M. Hennessy, *Distributed bisimulations*, Journal of the ACM, Volume 36 (1989), Number 4, Pp. 887–911.
15. P. Darondeau and P. Degano, *Causal trees*, in Automata, Languages and Programming (B. Rovan, Ed.), Volume 372, Springer-Verlag LNCS, 1989.
16. J. Davies and S. Schneider, *An introduction to timed CSP*, Technical Report Number 75, Oxford University Computer Laboratory, 1989.
17. P. Degano, R. De Nicola and U. Montanari, *A Partial Ordering Semantics for CCS*, Theoretical Computer Science, Volume 75 (1990), Pp. 223–262.
18. G. Ferrari, R. Gorrieri and U.Montanari, *Parametric laws for concurrency*, Manuscript, Dipartimento di Informatica, Università di Pisa, 1992.
19. G. Ferrari and U.Montanari, *Observing Time-Complexity of Concurrent Programs*, Manuscript, Dipartimento di Informatica, Università di Pisa, 1993.
20. R. van Glabbeek, *Comparative concurrency semantics and refinement of actions*, Ph.D. thesis, Vrije Universiteit te Amsterdam, 1990.
21. R. van Glabbeek, *A complete axiomatization for branching bisimulation congruence of finite–state behaviours*, in Mathematical Foundations of Computer Science (A. Borzyszkowski, Ed.), Volume 711, Springer-Verlag LNCS, 1993.
22. R. van Glabbeek and F. Vaandrager, *Petri net models for algebraic theories of concurrency*, in Proceedings of the PARLE conference, volume II (Parallel Languages) (J. de Bakker, A. Nijman and P. Treleaven Eds.), Volume 259, Springer-Verlag LNCS, 1987.
23. R. van Glabbeek and W. Weijland, *Branching time and abstraction in bisimulation semantics (extended abstract)*, in Information Processing '89 (G. Ritter Ed.), Pp. 613–618, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.
24. R. van Glabbeek and W. Weijland, *Refinement in branching time semantics*, in Proceedings of the AMAST Conference, Pp. 197–201, 1989. Also available as Report CS-R8922, CWI, Amsterdam, 1989.
25. J. Godskesen and K. Larsen, *Real–time calculi and expansion theorems*, in Proceedings of the 1st North American Process Algebra Workshop, 1992.
26. R. Gorrieri, *Refinement, atomicity and transactions for process description languages*, Ph.D. thesis, Dipartimento di Informatica, Università di Pisa, 1991, available as Technical Report TD 2/91.
27. R. Gorrieri and M. Roccetti, *Towards performance evaluation in process algebra*. Manuscript, Dipartimento di Matematica, Università di Bologna, 1993. To appear in the Proceedings of AMAST '93.
28. J.F. Groote and F.W. Vaandrager, *An efficient algorithm for branching bisimulation and stuttering equivalence*. Proceedings of the 17th ICALP, 1990, Pp. 626–638.
29. M. Hennessy, *Axiomatising Finite Concurrent Processes*, SIAM Journal of Computing, Volume 17 (1988), Number 5.

30. M. Hennessy and T. Regan, *A temporal process algebra*, Technical Report 2/90, Department of Computer Science, University of Sussex, 1990.
31. C. Hoare, *Communicating sequential processes*, International series on computer science, Prentice-Hall, 1985.
32. A. Jeffrey, *Timed Process Algebra* $\neq$ *Time* $\times$ *Process Algebra*, Technical Report Number 79, Programming Methodology Group, Chalmers University, 1991.
33. M. Joseph and A. Goswami, *Relating computation and time*, Technical Report RR 138, Department of Computer Science, University of Warwick, 1985.
34. L. Lamport, *On interprocess communication. part I: Basic formalism*, Distributed Computing, Volume 1 (1986), Pp. 77–85.
35. F. Mattern, *Virtual Time and Global States of Distributed Systems*, in Parallel and Distributed Algorithms, (M. Cosnard et al., Eds.), North-Holland, 1989.
36. A. Mazurkiewicz, *Traces, histories, graphs: Instances of a process monoid*, in Mathematical Foundations of Computer Science, Volume 176, Springer-Verlag LNCS, 1984.
37. R. Milner, *Communication and concurrency*, International series on computer science, Prentice Hall International, 1989.
38. R. Milner and F. Moller, *Unique decomposition of processes (note)*, Theoretical Computer Science, Volume 107 (1993), Number 2, Pp. 357–363.
39. F. Moller, *Axioms for concurrency*, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.
40. F. Moller and C. Tofts, *A temporal calculus of communicating systems*, in the Proceedings of Concur, Volume 459, Springer-Verlag LNCS, pp. 401–415, 1990.
41. D. Murphy, *Timed Process Algebra, Petri Nets, and Event Refinement, in Refinement* (J. Morris, Ed.), Springer-Verlag, 1991.
42. D. Murphy, *Intervals and actions in a timed process algebra*, Technical Report Arbeitspapiere der GMD 680, Gesellschaft für Mathematik und Dataverarbeitung, St. Augustin, 1992, presented at MFPS '92 and submitted to Theoretical Computer Science.
43. D. Murphy and D. Pitt, *Real–timed concurrent refineable behaviours*, in Proceedings of Formal Techniques in Real Time and Fault Tolerant Systems (J. Vytopil, Ed.), Volume 571, Springer-Verlag LNCS, 1992.
44. X. Nicollin and J. Sifakis, *The algebra of timed processes ATP: Theory and application*, Technical Report RT-C26, Laboratoire de Génie Informatique de Grenoble, 1990.
45. G. Plotkin, *A structural approach to operational semantics*, Technical Report DAIMI–FN–19, Computer Science Department, Århus University, 1981.
46. V. Sassone, M. Nielsen and G. Winskel, *A classification of models for concurrency*. In CONCUR (E. Best, Ed.), Volume 715, Springer-Verlag LNCS, 1993.
47. S. Schneider, *An operational semantics for timed CSP*, Information and Computation, Volume to appear (1992).
48. W. Vogler, *Failures Semantics based on Interval Semiwords is a congruence for Refinement*, Distributed Computing, Volume 4 (1991), Pp. 139–162.
49. W. Vogler, *Modular Construction and Partial Order Semantics of Petri Nets*, Volume 625, Springer-Verlag LNCS, 1993.
50. Wang Yi, *CCS + Time = an Interleaving Model for Real Time Systems*, in the Proceedings of ICALP, Springer-Verlag LNCS, 1991.