On the Ill–Timed but Well–Caused

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.

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 $a\,.\,\text{NIL} \;\|\; b\,.\,\text{NIL}$ to the choice between the alternative interleavings $(a\,.\,b\,.\,\text{NIL}) + (b\,.\,a\,.\,\text{NIL})$, 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 becomes even more problematic when we move to timed process algebra. Many of the classic interleaving algebras, such as BPA [6], 'theoretical' CSP [18] or CCS [22] have timed analogues ([5], [10] and [23] 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 $\langle a, b, c\rangle$, $\langle a, c, b\rangle$ and $\langle c, a, b\rangle$ with the process $(a\,.\,b\,.\,\text{NIL}) \;\|\; c\,.\,\text{NIL}$. 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 $\langle a@0, b@2, c@0\rangle$, $\langle a@0, c@0, b@2\rangle$ and $\langle c@0, a@0, b@2\rangle$. Many authors [5, 17, 23] 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

* *Postal addresses*: School of Cognitive and Computing Science, University of Sussex, Brighton BN1 9QH, England (LA) and Department of Computer Science, University of Birmingham, Birmingham B15 2TT, England (DVJM).

are often a great aid in manipulating expressions, establishing normal forms etc., rarely follow in this setting, as Godskesen and Larsen [13] 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.

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 $(a.b.\text{NIL}) \parallel c.\text{NIL}$, 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 [24]. It is not an 'industrial strength' process algebra, such as timed CSP, but it is expressive enough to demonstrate many of the features of timed concurrency without being so complex as to elude analysis. Before using cIPA in anger, though, we would have to add a number of features, such as timeouts, that are not discussed here.

An operational semantics is given for the algebra; this allows us to associate a timed transition system with a process. A notion of equivalence is then defined over our timed transition systems and shown to be a congruence of all of the algebra's operators. We show that an expansion theorem holds for the algebra, and that the independence of concurrent states is captured through timing information. This allows us to provide a complete axiomatisation of our notion of equivalence.

Our results here, we feel, show the usefulness of timed calculi; not only do they enable us to express timed behaviour, (which is important in itself [19],) but they also have semantic advantages, as we show, such as allowing us to code asynchronous transition systems as ordinary ones or simplifying the definition of action refinement. For reasons of brevity, all proofs are omitted; further details can be found in [2].

2. A Timed Process Algebra

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

2.1. Syntax

We will use a syntax derived from CCS [22]. In what follows, we assume 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), with $x = \overline{\overline{x}}$ for all $x \in A \cup \overline{A}$, writing ACT for $A \cup \overline{A}$.

We write $\mathbb{R}^+$ for the positive reals. The syntax of cIPA is then defined by

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

with a ranging over $\textsc{Act}$, $t \in \mathbb{R}^+$, 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 $a \in \textsc{Act}$ followed by the process P.
$\textsc{Wait}\, t\,.\,P$	A process that does nothing but wait 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.
$\textsc{Nil}$	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 $\textsc{Nil}$, writing for instance $a\,.\,b$ for $a\,.(b\,.\,\textsc{Nil})$. Moreover, we assume that . binds more strongly than $+$ or $||$, so that $a\,.\,b\,+\,c$ should be read as the process $(a\,.\,b\,.\,\textsc{Nil})\,+\,c\,.\,\textsc{Nil}$.

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 (the usual approach), 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 $\Delta : A \to \mathbb{R}^+$ which assigns durations to actions. The duration $\Delta(a)$ for any $a \in A$ will be assumed to be nonzero and constant over all occurrences of a. We extend Δ to $\textsc{Act}$ by defining $\Delta(\overline{a}) = \Delta(a)$.

Our approach of giving durations to actions is novel but not entirely new; Lamport, for instance, assumes nonatomicity [20] in his discussions of distributed systems, while related process algebraic work with explicit starts and finishes is due to Aceto and Hennessy [3], [16]. Both van Glabbeek (in his definition of ST-bisimulation [12]) and Gorrieri [14] imply that durational information is necessary in the consideration of action refinement and hence in properly structuring the descriptions of systems; indeed, our calculus has a particularly clean notion of action refinement; see section 3.4.

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... units of time after a has started. The urgency of cIPA allows us to define the set of all times a process without 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.)

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

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

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 [26], 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, g will range over $\text{Act} \cup \{\tau\}$.

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

$$\frac{s_i \xrightarrow{g_i} s_i' \qquad (i \in I)}{u \xrightarrow{g} u'}$$

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

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

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 $E \subset (\text{Act} \cup \{\tau\}) \times \mathbb{R}^+ \times \mathbb{R}^+$ (together with a duration map $\Delta : \text{Act} \rightarrow \mathbb{R}^+$), a relation $\rightarrow : S \times E \times S$ and an initial configuration s_0. We usually write $s \xrightarrow[\delta]{g@t} u$ rather than $(s,(g,t,\delta),u) \in \rightarrow$, and require that τ is the only action whose duration can vary: $s \xrightarrow[\delta]{g@t} u$ and $s' \xrightarrow[\delta']{g@t'} u'$ implies $\delta = \delta'$ or $g = \tau$.

For the sake of convenience, following [12, Chapter 3] we shall only consider *root unwound* ATTSs, i.e. ATTSs with no incoming edges at the root.

2.5. Configurations

Consider the timed transition rule above: 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 in timed transition rules of knowing the time of occurrence of an action. Thus we define the set $C(\text{cIPA})$ 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 $\mathcal{C}$(cIPA) — indeed we have identified cIPA with a subset of the set of *generators* of $\mathcal{C}$(cIPA). The configuration $(P\ t) \parallel (P'\ t')$ 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.

For the sake of simplicity, we shall follow the example of [9] and consider configurations in canonical form with respect to the operators $\_\ t$. In particular, it will be assumed that the operators $\_\,t$ distribute over the nondeterministic choice, restriction and parallel composition. Formally, let $\equiv$ denote the least congruence over these operators which satisfies the following axioms:

$$(P+Q)\,t \;=\; P\,t+Q\,t \qquad\qquad (P\backslash C)\,t \;=\; (P\,t)\backslash C$$
$$(P \parallel Q)\,t \;=\; (P\,t) \parallel (Q\,t)$$

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

$$s \;::=\; \text{NIL}\,t \mid (a\,.P)\,t \mid (\text{WAIT}\,t'\,.P)\,t \mid s+s \mid s\backslash C \mid s \parallel s$$

such that s is always $\equiv$ to a canonical term. In what follows, $\mathcal{C}$(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 $s \xrightarrow[\delta]{g@t} s'$ which means that starting from the configuration s, the configuration s' is reachable by the action g 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.

2.6. ATTS Semantics

The ATTS associated with a cIPA expression is defined by the rules in displays 1 and 2. Our timed semantics will be rather different from the usual ones in the literature [5, 17, 23, 27] 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. (This remains true even when recursion is added to the calculus; see section 5.1.) These two features mean that all processes in our calculus will be *implementable* [19, 25].

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 $\mathtt{wait}\; t \rightarrow P$. This is because we think the moment when choices are made 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

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

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 [17], 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 despite choices being made at different times in the two cases.)

$$\text{ACT}\,\frac{}{(a\,.\,P)\;t \xrightarrow[\Delta(a)]{a@t} P\;(t+\Delta(a))} \qquad \text{WAIT}\,\frac{}{(\text{WAIT}\,t'\,.\,P)\;t \xrightarrow[t']{\tau@t} P\;(t+t')}$$

$$\text{HC1}\,\frac{s \xrightarrow[\delta]{a@t} s'}{(s\backslash C) \xrightarrow[\delta]{a@t} (s'\backslash C)}\;a,\bar{a} \notin C \qquad \text{HC2}\,\frac{s \xrightarrow[\delta]{\tau@t} s'}{(s\backslash C) \xrightarrow[\delta]{\tau@t} (s'\backslash C)}$$

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

DISPLAY 1. The Timed Operational Semantics for cIPA without parallelism.

The rules for restriction are HC1 and HC2 : restriction just stops the restricted actions from happening; τs can always proceed since $C \subseteq A$, and $\tau \notin A$.

$$\text{PAL}\,\frac{s_1 \xrightarrow[\delta]{g@t} s_1'}{s_1 \,||\, s_2 \xrightarrow[\delta]{g@t} s_1' \,||\, s_2} \qquad \text{PAR}\,\frac{s_1 \xrightarrow[\delta]{g@t} s_1'}{s_2 \,||\, s_1 \xrightarrow[\delta]{g@t} s_2 \,||\, s_1'}$$

$$\text{SYNC}\,\frac{s_1 \xrightarrow[\delta]{a@t} s_1' \quad s_2 \xrightarrow[\delta]{\bar{a}@t} s_2'}{s_1 \,||\, s_2 \xrightarrow[\delta]{\tau@t} s_1' \,||\, s_2'}$$

DISPLAY 2. The timed operational semantics of parallelism in cIPA.

The rules for parallelism are PAL, PAR and SYNC. The idea of a configuration $P\,t \,||\, Q\,t'$ is that t and t' are the local clocks at P and Q. Notice that the two clocks never interact during the execution of $P \,||\, Q$, 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). (See [15] for a related timed model with a looser notion of synchronisation.) 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 conceptual ones, and 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 [20].

Consider the process $(a.b \mid\mid \bar{b}.c)\backslash\{b\}$. In the presence of rule SYNC, this process will perform an a-action at time 0 and then deadlock, because $b@\Delta(a)$ and $\bar{b}@0$ 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.

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 ATTSs 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 [12].

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 $s \xrightarrow[\delta]{g@t} s'$ then exactly one of the following applies

$g = \tau$. 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 $s'\ R\ u''$.

$g = a$. 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 are RBB if their meanings are.

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 $a \parallel b$. This generates the traditional diamond with paths

$$(a \parallel b)\, 0 \xrightarrow[\Delta(a)]{a@0} (\text{NIL}\,\Delta(a)) \parallel (b\,0) \xrightarrow[\Delta(b)]{b@0} (\text{NIL}\,\Delta(a)) \parallel (\text{NIL}\,\Delta(b))$$

$$(a \parallel b)\, 0 \xrightarrow[\Delta(b)]{b@0} (a\,0) \parallel (\text{NIL}\,\Delta(b)) \xrightarrow[\Delta(a)]{a@0} (\text{NIL}\,\Delta(a)) \parallel (\text{NIL}\,\Delta(b))$$

In contrast, we have the two transitions

$$(a\,.\,b\ +\ b\,.\,a)\, 0 \xrightarrow[\Delta(a)]{a@0} b\,\Delta(a) \xrightarrow[\Delta(b)]{b@\Delta(a)} \text{NIL}\,(\Delta(a)+\Delta(b))$$

$$(a\,.\,b\ +\ b\,.\,a)\, 0 \xrightarrow[\Delta(b)]{b@0} a\,\Delta(b) \xrightarrow[\Delta(a)]{a@\Delta(b)} \text{NIL}\,(\Delta(a)+\Delta(b))$$

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

Restriction. Consider $P = (a\,.\,b \parallel \overline{a})\backslash\{a\}$ and $Q = (c\,.\,b \parallel \overline{c})\backslash\{c\}$. 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, $\approx$ is even finer than strong bisimulation over semantically finite sequential processes, which is not surprising given the rôle played by timing information in cIPA.

NonLaw. It is interesting that none of the τ laws in [12] immediately generalises to our setting. For instance, the restriction example above generalises to $a\,.\,\text{WAIT}\,t\,.\,b \not\approx a\,.\,b$. We also have $\text{WAIT}\,t\,.\,a \not\approx \text{WAIT}\,t\,.\,a\ +\ a$ since the latter can begin a earlier than the former.

Note, however, that $a\,.\,\text{WAIT}\,t \approx a$ 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. There are some nontrivial relationships between processes with τs, for instance, $a\,.(\text{WAIT}\,t\,.\,\text{WAIT}\,t') = a\,.\,\text{WAIT}\,(t+t')$. Indeed, it is easy to see that $a\,.\,\text{WAIT}\,t \approx a\,.\,\text{WAIT}\,t'$ and $\text{WAIT}\,t \approx \text{WAIT}\,t'$ for all $t, t' \in \mathbb{R}^+$. Moreover, we have that $a\,.(\text{WAIT}\,t \parallel b) \approx a\,.\,b$.

3.2. Compositionality and Timing

We will now show that rooted branching bisimulation is a congruence and begin to investigate the relationship between timing and trace order.

Proposition 5. Rooted branching bisimulation is a congruence with respect to all the cIPA operations.

Suppose that we have the following path in an ATTS, G: $s \xrightarrow[\delta_1]{g_1@t_1} s' \xrightarrow[\delta_2]{g_2@t_2} s''$. Since our actions are urgent, – they happen as quickly as they can, – we expect g_2 to have started immediately after g_1 has ended, and so we expect $t_2 = t_1 + \delta_1$. 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 ($t_2 < t_1 + \delta_1$). Our next proposition shows that ill–timed paths precisely arise through concurrency:

Proposition 6. Suppose that P is a cIPA process without restriction. Then $[\![P]\!]$ contains an ill–timed path iff P has a subterm of the form $Q \parallel R$ with neither Q nor R equivalent to NIL.

3.3. Independency in ATTSs

The concept of independency has been proposed by several authors, notably Bednarczyk [7], and Mazurkiewicz [21] 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 $s \xrightarrow[\delta_1]{g_1 @ t_1} s_1 \xrightarrow[\delta_2]{g_2 @ t_2} u$ and $t_2 < t_1 + \delta_1$; then g_1 must be independent from g_2 since it began before the other terminated. Thus in particular the path $s \xrightarrow[\delta_2]{g_2 @ t_2} u_1 \xrightarrow[\delta_1]{g_1 @ t_1} u$ should also be possible; this is called the *diamond property* by Bednàrczyk, and motivates the following.

Definition 7. Suppose that $G = (S, E, \rightarrow, s_0)$ 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: $s \xrightarrow[\delta_1]{g_1 @ t_1} s_1 \xrightarrow[\delta_2]{g_2 @ t_2} u$ and $t_2 \neq t_1 + \delta_1$ implies that there exists a $s'_1 \in S$ such that $s \xrightarrow[\delta_2]{g_2 @ t_2} s'_1 \xrightarrow[\delta_1]{g_1 @ t_1} u$.

For a cIPA process all time happens 'during' a transition, so there are no gaps in time; this is an *urgency* property and leads to:

Definition 8. Suppose that $G = (S, E, \rightarrow, s_0)$ 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: $s_0 \xrightarrow[\delta]{g @ t} u \quad \Longrightarrow \quad t = 0$.

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

Proposition 9. If P is a cIPA process, then $[\![P]\!]$ is a timeful, well-caused, finite ATTS.

3.4. 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 [12, §3.6].

We shall, following Gorrieri [14], think of action refinement as a tool for structuring the meanings of complex 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 10. A process P is a *valid refinement* of an action a, written $P\,\mathbf{refines}\,a$, 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;

[†] In fact, this observation forms the basis of a translation from (a certain class of) well-caused ATTSs into acyclic Asynchronous Transition Systems (ATSs), the proof being given in [2].

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.

Definition 11 (Semantic Substitution). Let $F : A \rightarrow \text{cIPA}$ be a mapping from actions to cIPA processes with the property that

$$F(a) = P \quad\Longrightarrow\quad P \textbf{ refines } a$$

Then, we call F a *semantic substitution*, and for an 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 $[\![F(a)]\!]$. Identify s with the root node of $F(a)_i$ and s' with the end nodes of $F(a)_i$, and remove the edge $s \xrightarrow[\delta]{a@t} s'$.

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 12. Suppose that G, H are ATTSs 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 $[\![F(a)]\!]$ 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 $[\![F(a)]\!]$.

Theorem 13. If $R : G \approx H$ is a RBB between ATTSs G and H, and F is a semantic substitution, then $F(R)$ is a branching bisimulation between $F(G)$ and $F(H)$.

4. 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 $\mathcal{C}(\text{cIPA})$. 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.

There is, after van Glabbeek and Weijland [12], a standard way of axiomatizing rooted branching bisimulation-like relations over ordinary, finite, acyclic labeled transition systems. The application of their method 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. In order to apply it, we extend the language $\mathcal{C}(\text{cIPA})$ to $\mathcal{EC}(\text{cIPA})$, where $\mathcal{EC}(\text{cIPA})$ is built up just as $\mathcal{C}(\text{cIPA})$ is, but 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 $\mathcal{C}(\text{cIPA})$ has been extended by allowing prefixing operators of the form $\alpha : \_$, for $\alpha \in E$. The language $\mathcal{EC}(\text{cIPA})$ 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 $\mathcal{EC}(\text{cIPA})$ inherits the structural congruence $\equiv$ from its sublanguage $\mathcal{C}(\text{cIPA})$, and in what follows $\mathcal{EC}(\text{cIPA})$ will always be considered modulo $\equiv$. We shall use $s, u, w, s', \ldots$ to range over $\mathcal{EC}(\text{cIPA})$ and α, β to range over E. The operational semantics for $\mathcal{EC}(\text{cIPA})$ is obtained by extending the rules in displays 1 and 2 with the axiom

$$\text{PRE} \quad \frac{}{(g, t, \delta) : s \xrightarrow[\delta]{g@t} s}$$

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

Proposition 14. $\approx$ is a congruence over $\mathcal{EC}(\text{cIPA})$.

Definition 15 (Sumforms). The set of *sumforms* over E is the least subset of $\mathcal{EC}(\text{cIPA})$ such that the following hold:

(1) NIL is a sumform (recall that we identify NIL with NIL 0);
(2) if $\alpha \in E$ and s is a sumform then $\alpha : s$ is a sumform;
(3) if s and u are sumforms, so is $s + u$.

In order to give a complete axiomatization for $\approx$ over $\mathcal{EC}(\text{cIPA})$ (and, consequently, over cIPA), it will be sufficient to devise a set of axioms which allow us to reduce terms in $\mathcal{EC}(\text{cIPA})$ 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 $\mathcal{EC}(\text{cIPA})$ given in display 3 together with the following TAU axiom over E:

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

Given this axiom, we shall abbreviate all timed, durationful τ-actions to τ. The equivalence relation over E generated by the 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:

$$\text{SUB} \qquad \alpha \equiv_E \beta \text{ and } s = u \text{ implies } \alpha : s = \beta : u$$

In what follows, for all $s, u \in \mathcal{EC}(\text{cIPA})$, we shall write $s =_\mathcal{C} u$ iff the equality $s = u$ can be derived using the equations in display 3 and the rule SUB using the rules of equational logic.

$$\begin{array}{rcll}
s+u &=& u+s & (A1)\\
(s+u)+w &=& s+(u+w) & (A2)\\
s+s &=& s & (A3)\\
s+\mathrm{NIL} &=& s & (A4)\\[6pt]
\alpha:(\tau:(s+u)+s) &=& \alpha:(s+u) & (H)\\[6pt]
(P+Q)\,t &=& P\,t+Q\,t & (S1)\\
(P\backslash C)\,t &=& (P\,t)\backslash C & (S2)\\
(P\,||\,Q)\,t &=& (P\,t)\,||\,(Q\,t) & (S3)\\[6pt]
(a\,.\,P)\,t &=& (a,t,\Delta(a)):(P\,(t+\Delta(a))) & (R1)\\
(\mathrm{WAIT}\,t'\,.\,P)\,t &=& \tau:(P\,(t+t')) & (R2)\\
\mathrm{NIL}\,t &=& \mathrm{NIL} & (R3)\\
(s+u)\backslash C &=& s\backslash C+u\backslash C & (R4)\\
((g,t,\delta):s)\backslash C &=& \mathrm{NIL} \qquad \text{if } g\in C\cup\overline{C} & (R5)\\
((g,t,\delta):s)\backslash C &=& (g,t,\delta):s\backslash C \qquad \text{if } g\notin C\cup\overline{C} & (R6)
\end{array}$$

$$\left(\sum_{i\in I}(g_i,t_i,\delta_i):s_i\right)\,\Bigg|\Bigg|\,\left(\sum_{j\in J}(g'_j,t'_j,\delta'_j):u_j\right) = \qquad (Int)$$

$$\sum_{i\in I}(g_i,t_i,\delta_i):\left(s_i\,\Bigg|\Bigg|\left(\sum_{j\in J}(g'_j,t'_j,\delta'_j):u_j\right)\right)$$

$$+\sum_{j\in J}(g'_j,t'_j,\delta'_j):\left(\left(\sum_{i\in I}(g_i,t_i,\delta_i):s_i\right)\Bigg|\Bigg|\,u_j\right)+\sum_{(i,j):\,g_i=\overline{g_j},t_i=t_j}\tau:(s_i\,||\,u_j)$$

DISPLAY 3. Equations over configurations.

A few comments on the axioms in display 3 are now in order. Axioms (A1)–(A4) and (H) are all that are needed to completely axiomatize rooted branching bisimulation over E-labeled finite trees [12]. Axioms (S1)–(S3) are the structural axioms over the set of generators of $\mathcal{C}$(cIPA) and $\mathcal{EC}$(cIPA). These axioms are used to reduce configurations to finite E-labeled trees together with axioms (R1)–(R6) and (*Int*).

Other authors, notably Ferrari et al. [11], 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.[‡]

Theorem 16. For all $s,u\in\mathcal{EC}$(cIPA), $s\approx u$ iff $s=_c u$. In particular, for all cIPA processes P,Q, $P\approx Q$ iff $P\,0=_c Q\,0$.

[‡] We could treat other notions of synchronisation by altering the equation (*Int*); for instance, by allowing actions which did not happen at exactly the same time to synchronise. Our technical treatment would then be similar, although whether $\approx$ is still a congruence is problematic; see [15].

5. Concluding Remarks

We have defined a timed process algebra with an interleaving semantics where concurrency is expressed through simultaneity (proposition 6). Event refinement goes through smoothly (theorem 13), and we can characterise equivalence of structures algebraically (theorem 16). Moreover, the fact that that our semantic structures are just labeled trees means that much of interleaving concurrency theory can be reused in our setting; indeed, our complete axiomatisation is an example of this. We hope that this work will aid the search for tractable, expressive formalisms to model the timed behaviour of distributed systems.

To end the paper, we discuss extensions of the work presented and make some more detailed comparisons with related work.

5.1. Recursion in cIPA

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 means of equations $X \stackrel{\text{def}}{=} P$, 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

$$\textsc{Rec}\ \frac{P\,t \xrightarrow[\delta]{g@t} s}{X\,t \xrightarrow[\delta]{g@t} s}\ (X \stackrel{\text{def}}{=} P)$$

All the results presented in the paper, apart from the complete axiomatization,—which is intrinsically limited to finite processes,—lift to cIPA with recursive definitions.

5.2. Comparison

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. We indicate the main approaches here, and their relationship to our own, making no claim of completeness. As an aid in the following discussion, we shall make use of the standard example in the literature, namely the processes $P = a \,||\, b$ and $Q = a\,.\,b + b\,.\,a$.

- Boudol et al. [8] 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.
- Aceto and Hennessy [16, 3] 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 [9] 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; the use of timing and duration information enables us to *discover* which states are independent. Thus our ATTSs, like asynchronous transition systems [7, 28], 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. The idea that a location is just somewhere a local clock can be is one that we hope to explore further.

A proper study of the formal relationships between the congruence investigated in this paper and the noninterleaving congruences mentioned above is hindered by the fact that we have chosen to model $\approx$ upon branching bisimulation rather than ordinary weak bisimulation; to undertake a proper study, we should relate equivalences of the same power with respect to branching. However, for "concrete" processes (i.e. processes without internal transitions) built from the operators NIL, action-prefixing, choice and parallelism (without synchronization) we can prove the following result:

Theorem 17. For concrete processes, $\approx$ coincides with causal bisimulation equivalence [9], location equivalence [8] and ST-bisimulation equivalence [12].

For processes with internal transitions, $\approx$ distinguishes more purely sequential processes than strong bisimulation equivalence (see the restriction example above) 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. [2] contains a more comprehensive analysis of the relationship of $\approx$ to other equivalences in the literature.

5.3. Acknowledgments

The authors would like to thank Gian Luigi Ferrari, Vladimiro Sassone and the anonymous referees 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, D. Murphy, *Timing and Causality in Process Algebra*. In preparation.
3. L. Aceto and M. Hennessy, *Towards action-refinement in process algebra*, Information and Computation, Volume 103, Number 2 (1993).
4. T. Axford, *Concurrent Programming: Fundamental Techniques for Real-Time and Parallel Software Design*, Wiley, 1989
5. J. Baeten and J.A. Bergstra, *Real time process algebra*, Formal Aspects of Computing, Volume 3 (1991), Number 2, Pp. 142–188.
6. J. Baeten and W. Weijland, *Process algebra*, Cambridge Tracts in Theoretical Computer Science, Volume 18, Cambridge University Press, 1990.
7. M. Bednarczyk, *Categories of asynchronous systems*, Ph.D. thesis, Department of Computer Science, University of Sussex, 1987, available as Technical Report Number 3/87.

8. G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn, *A theory of processes with locality*, Technical Report 13/91, Department of Computer Science, University of Sussex, 1991.
9. P. Darondeau and P. Degano, *Causal trees*, in Automata, Languages and Programming (B. Rovan, Ed.), Volume 372, Springer-Verlag LNCS, 1989.
10. J. Davies and S. Schneider, *An introduction to timed CSP*, Technical Report Number 75, Oxford University Computer Laboratory, 1989.
11. G. Ferrari, R. Gorrieri, and U.Montanari, *Parametric laws for concurrency*, manuscript, Dipartimento di Informatica, Università di Pisa, 1992.
12. R. van Glabbeek, *Comparative concurrency semantics and refinement of actions*, Ph.D. thesis, Vrije Universiteit te Amsterdam, 1990.
13. J. Godskesen and K. Larsen, *Real–time calculi and expansion theorems*, in Proceedings of the 1st North American Process Algebra Workshop, 1992.
14. 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.
15. R. Gorrieri and M. Roccetti, *Towards performance evaluation in process algebra*. To appear in the proceedings of AMAST 1993.
16. M. Hennessy, *Axiomatising Finite Concurrent Processes*, SIAM Journal of Computing, Volume 17, Number 5, 1988.
17. M. Hennessy and T. Regan, *A temporal process algebra*, Technical Report 2/90, Department of Computer Science, University of Sussex, 1990.
18. C. Hoare, *Communicating sequential processes*, International series on computer science, Prentice-Hall, 1985.
19. M. Joseph and A. Goswami, *Relating computation and time*, Technical Report RR 138, Department of Computer Science, University of Warwick, 1985.
20. L. Lamport, *On interprocess communication. Part I: Basic formalism*, Distributed Computing, Volume 1 (1986), Pp. 77–85.
21. A. Mazurkiewicz, *Traces, histories, graphs: Instances of a process monoid*, in Mathematical Foundations of Computer Science, Volume 176, Springer-Verlag LNCS, 1984.
22. R. Milner, *Communication and concurrency*, International series on computer science, Prentice Hall International, 1989.
23. 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.
24. 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.
25. 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.
26. G. Plotkin, *A structural approach to operational semantics*, Technical Report DAIMI–FN–19, Computer Science Department, Århus University, 1981.
27. S. Schneider, *An operational semantics for timed CSP*, manuscript, Programming Research Group, Oxford University. To appear in Information and Computation.
28. V. Sassone, M. Nielsen and G. Winskel, *A classification of models for concurrency*, this volume.