# Timing and Causality in Process Algebra

Luca Aceto and David Murphy

*University of Sussex and University of Birmingham*

with the process $(a\,.\,b\,.\,\textsc{Nil}) \;||\; c\,.\,\textsc{Nil}$. If we add timing, writing $a$

## 2.1. Syntax

We will use a syntax derived from both CSP [29] and CCS [35], assuming as given some set of actions, $A$, with $\tau$ (which will be used for internal actions of a process) not in $A$. We also assume a bijection $\overline{\phantom{a}} : A \rightarrow \overline{A}$ (giving the complementary action of $a$), extend $\overline{\phantom{a}}$ 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

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

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 $X$.)

$$\begin{array}{rclrcl}
\mathbf{dur}(\textit{N}\scriptstyle{IL}\textstyle) & = & \{0\} & \mathbf{dur}(\textit{W}\scriptstyle{AIT}\textstyle\, t \,.\, P) & = & \{t' + t \mid t' \in \mathbf{dur}(P)\} \\
\mathbf{dur}(P + Q) & = & \mathbf{dur}(P) \cup \mathbf{dur}(Q) & \mathbf{dur}(a \,.\, P) & = & \{t' + \Delta(a) \mid t' \in \mathbf{dur}(P)\}
\end{array}$$

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

*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 [18, 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 y.

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

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

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 [3]. Moreover, our clocks are not supposed to model all of the features of physical clocks in actual implementations, so abstracting awa0Td(the)Tj19.6

duration: if we have $s\ R\ u$

Note, however, that $a\,.\,\textit{Wait}\ t \;\approx\; a$ 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 $\tau$s, e.g.

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

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

## 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

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

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

**Lemma 6.** For all $s \in \mathcal{C}(\textit{cIpa})$ 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 $s' \in \mathcal{C}(\textit{cIpa})$ 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 *cIpa* processes $P$ and times $t \in \mathbb{R}^+$, $t \in \mathbf{dur}(P)$ iff there exists $s \in \mathcal{C}(\textit{cIpa})$ such that $P \rightarrow^\star s \not\rightarrow$ and $\mathbf{maxtime}(s) = t$.

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

We can now tackle the compositionality of 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 $a \in \textit{Act}$,

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

*Proof.* This is an extension to *cIpa* of van Glabbeek and Weijland's for BPA [21]. The only cases which depart slightly from the standard proof are those for action and $\textit{Wait}\,t$ 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

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

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

## 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 $\Delta$. For instance, an equality like

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

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 $\delta$ to each action; using such a duration function, all the transitions between *WAIT* $t$–free configurations will have duration $\delta$. Furthermore, we write $\xrightarrow[\delta]{\mu@t}$ for the

$\mu$

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

**Proposition 19.** If $P$ is a *cIpa* process, then $[\![P]\!]$ 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 $s \in \mathcal{C}(cI\textsc{pa})$,

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

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 [22] and [18, §3.6].

We shall, following Gorrieri [25], 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 [illegible]
some speeds and fail at faster rates [3]. Note that we do not allow v[19.20T516.08018ai4sesomeis [illegible]

Now suppose that $\Delta(b) = \Delta(a) + \Delta(c)$ and consider the mapping $\quad : A \rightarrow$ *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

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 $(H)$ we find a path $u \Rightarrow u_1 \xrightarrow[\delta]{\mu@t} u'$ s.t. $s\ \ (R)\ u_1$ and $s'\ \ (R)\ u'$.

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

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

□

## 5. An Algebraic Characterization of ≈

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 [18, 19], 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:

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

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 $\alpha, \beta$ to range over $E$. The operational semantics for $\mathcal{EC}(cIPA)$ is obtained by extending the rules in displays 1 and 2 displaa

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

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

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

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

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 $\mathcal{EC}(c\textit{Ipa})$ (and, consequently, over *cIpa*

We can now state the promised completeness theorem:

**Theorem 26.** For all $s, u \in \mathcal{EC}(c\textit{IPA})$, $s \approx u$ iff $s =_{\mathcal{C}} u$. In particular, for all $c\textit{IPA}$ 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 $s, u \in \mathcal{EC}(c\textit{IPA})$, $s =_{\mathcal{C}} u$ implies $s \approx u$;

Completeness for sumforms [18, 19]: 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 $s \in \mathcal{EC}(c\textit{IPA})$ is provably equal to a sumform, i.e. for each $s \in \mathcal{EC}(c\textit{IPA})$ 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 [35].

The completeness of axioms (A1)-(A4) and (H) for branching bisimulation over finite trees has been proven by van Glabbeek and Weijland in [18, 19].

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. [17], 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 treat the loose notion of timed synchronisation given in [24]. It should be noted, however [*op cit.*], that RBB would not then be a congruence, so such modifications are not always wholly propitious.

## 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 $P = a \mid\mid b$ and $Q = a\,.\,b + b\,.\,a$.

- Boudol et al. [11] 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, 27] 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.

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

Display 4.

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

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

location equivalence. In fact, regardless of the duration of actions $a$ and $b$, it would identify the processes

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

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 $P = (a\,.\,b \mid\mid$

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 $\kappa \subset E'$

## 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

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

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

The data-gatherer, then, will be responsible for doing a *Start*1 with the correct $t_1'$, which will ensure that the sensor does a *Read*1 when required. It must also be ready $t_1' + \Delta(\textit{Read}1)$ units of time after finishing the *Start*1 to do a $\overline{\textit{Write1}}$, allowing the sensor to transfer data back to it. Finally, it *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

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

(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 *Read*1 and *Read*2 have deadlines $d_1$ and $d_2$ respectively, i.e. *Readi* must happen at $t = d_i$. Clearly, then, we must have the deadline equation

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

in order for the deadlines to be met.

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

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

and

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

overlap, then a purely sequential implementation is not possible, as the data-gatherer will need to do both a $\overline{\textit{Write}1}$ and a $\overline{\textit{Write}2}$ 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

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

(which is eminently reasonable, as the *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).

arithmetic (and t[redacted]s that correct functioning depends on the behaviour shown in display 5.

| Ti[redacted] | [redacted]Action | Sensor2 Action | Data-Gatherer Action |
|---|---|---|---|
| [redacted] | [redacted]ng | Waiting | Waiting |
| $t$[redacted] | [redacted]ng | $\overline{\textit{Start}2}$ | $\textit{Start}2$ |
| $t_2 + \Delta($[redacted] | [redacted]ng$^{\dagger}$ | Waiting | Waiting |
| $t$[redacted] | [redacted]$\overline{T1}$ | Waiting | $\textit{Start}1$ |
| $t_1 + \Delta($[redacted] | [redacted]ng | Waiting | Waiting |
| $d$[redacted] | [redacted]$D1$ | Waiting | Waiting |
| $d_1 + \Delta$[redacted] | [redacted]$\overline{E1}$ | ? | $\textit{Write}1$ |
| $d$[redacted] | [redacted] | $\textit{Read}2$ | ? |
| $d_2 + \Delta$[redacted] | [redacted] | $\overline{\textit{Write}2}$ | $\textit{Write}2$ |

[redacted]ming constraints on the Data-Gatherer.

The ? occur [redacted]know where warming up the second sensor occurs relative to taking the firs[redacted]is is a scheduling issue that depends on the relationship between the $d_i$ an[redacted]happen. But the time between being activated and taking a measurement is a [redacted]hysical) property of each sensor; call it $warm_i$. $_i$. $_i$

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

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

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 *Sensori*s and $t_i$s, 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

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

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

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

7. M. Bednarczyk,

31. M. Joseph and A. Goswami, *Relating computation and time*, Technical Report RR 138, Department of Computer Science, University of Warwick, 1985.
32. L. Lamport, *On interprocess communication. part I: Basic formalism*, Distributed Computing, Volume 1 (1986), Pp. 77–85.
33. F. Mattern, *Virtual Time and Global States of Distributed Systems*, in Parallel and Distributed Algorithms, (M. Cosnard et al., Eds.), North-Holland, 1989.
34. A. Mazurkiewicz, *Traces, histories, graphs: Instances of a process monoid*, in Mathematical Foundations of Computer Science, Volume 176, Springer-Verlag LNCS, 1984.
35. R. Milner, *Communication and concurrency*, International series on computer science, Prentice Hall International, 1989.
36. R. Milner and F. Moller, *Unique decomposition of processes (note)*, Theoretical Computer Science, Volume 107 (1993), Number 2, Pp. 357–363.
37. F. Moller, *Axioms for concurrency*, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.
38. 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.
39. 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.
40. 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.
41. 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.
42. G. Plotkin, *A structural approach to operational semantics*, Technical Report DAIMI–FN–19, Computer Science Department, Århus University, 1981.
43. V. Sassone, M. Nielsen and G. Winskel, *A hierarchy of models for concurrency*. To appear as a DAIMI Technical Report, Århus University, 1993.
44. S. Schneider, *An operational semantics for timed CSP*, Information and Computation, Volume to appear (1992).
45. Wang Yi, *CCS + Time = an Interleaving Model for Real Time Systems*, in the Proceedings of ICALP, Springer-Verlag LNCS, 1991.