A Real Time Process Logic

J.C.M. Baeten[1]
Department of Computing Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands. (josb@win.tue.nl)

J.A. Bergstra[1,2]
Programming Research Group, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
and
Department of Philosophy, Utrecht University,
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands. (janb@fwi.uva.nl)

R.N.Bol
Department of Computing Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands. (bol@win.tue.nl)

Abstract

Systems can be described at various levels of abstraction: automata, processes and behavior. In this paper, we take the ready trace set as a description of the behavior of a process and we present a ready trace model of real time process algebra. We argue that, especially in the real time case, properties of ready trace sets are best formulated in a dedicated logic (as opposed to describing them in an enriched process notation, such as ACP_τ). We present the syntax and semantics of a logic that could serve this purpose and we apply it to study the existence of so-called coordinated attack protocols. A connection is made with the metric temporal logic of Koymans. This paper is an abbreviated version of [BABB93].

1. Introduction.

Computing systems can be described at various levels of abstraction: automata, processes and behavior. Trace theory is the name for a collection of semantic models for the design, description, and analysis of systems, in which the behavior of a system is described by a set of execution traces, or initial fragments of such traces. A trace encodes a possible behavior of the system by a, finite or infinite, sequence of atomic actions, possibly augmented with additional information. In this paper we concentrate on *real time ready traces* (see Section 1.2). By defining operators on trace sets, we obtain algebras of process ready trace sets that are models of certain versions of ACP [BEK84, BAB91].

We argue that, especially in the real time case, properties of ready trace sets are best formulated in a dedicated logic. An alternative would be the description of a property by a characteristic process. This alternative is attractive in the untimed case, using a hiding operator (i.e., an operator that abstracts from internal actions). However, in the real time case, defining a good hiding mechanism is hard, if at all possible: it is usually desirable to hide a part of the timing information, but not all of it.

We propose a behavioral property logic called RdTrL (Ready Trace Logic). Its syntax and semantics are closely related to ready trace theory. We compare the expressivity of RdTrL with the metric temporal logic of KOYMANS [KOY89a,b]. We demonstrate how the logic can help in studying the (non-)existence of coordinated attack protocols (as was done in [HAM90] using epistemic logic). We conclude by listing some shortcomings of the logics, thus pointing out topics for further research.

[1] This author received partial support from ESPRIT Basic Research Action 7166, CONCUR2.
[2] This author received partial support from ESPRIT Basic Research Action 6454, CONFER.

1.1. Conceptual Abstraction.

Systems are described at various levels of abstraction. The most basic distinction, on which we elaborate in the rest of this section, is the distinction between automata, processes, and behavior. We shall call this the *conceptual abstraction hierarchy*. Its levels are comparable to the Nets, Terms, and Formulas of [OLD91].

Within each level in this hierarchy, there are different dimensions and levels of *technical abstraction*. As an example, one of these dimensions is the treatment of time, where we can distinguish (from low to high abstraction): real space/time, real time, discrete time, and symbolic (untimed). Where necessary, we can identify intermediate levels as well. Without aiming at completeness, we list the following dimensions of technical abstraction:

- treatment of time,
- treatment of silent steps,
- treatment of divergence and fairness,
- probabilistic vs. non-probabilistic,
- choice abstraction (as described in [vGL90] for processes).

An *automaton* or transition system is a class of states equipped with a transition relation. Transitions are labelled with so-called actions. Labels (attributes) of states are called signals. Automata represent the lowest level of (conceptual) abstraction in system description needed in this document. Automata exist in many levels of (technical) abstraction: with and without roots, termination states, deadlocks, time stamps, probability assignments, fairness conditions on paths etc. A format that allows one to present automata is often called a *program notation*. Operational semantics assigns an automaton to a program. Of course, the program notation reflects the technical level of abstraction at which the automata are to be treated.

A *process* is an abstraction of an automaton. At this level of conceptual abstraction, information about states is hidden, wheras information about actions is kept. Important is that a process has (in general) an operational semantics and allows simulation (but not necessarily implementation). A format that allows one to describe processes can be called a *process notation*. In general, one expects the existence of an algorithm that returns a simulator, when given a term in process notation. Processes are usually shaped as equivalence classes of states of automata.

A *behavior* is a (conceptual) abstraction of a process. Seen as an abstraction of an automaton, it hides even more information about the state space than a process. The usual shape of a behavior is a collection (satisfying certain natural closure conditions) of traces of (system) paths in an automaton, all starting in the same root. The difference between a path and a trace is that a path contains all information of the states that are visited, whereas a trace will abstract (to some extent) from this intermediate state information.

A *behavioral property* is a collection of traces or a logical description of such a collection in some appropriate logical format. The main difference between a behavior and a property is that a property need not satisfy any closure conditions. A format (logic) that allows one to specify properties may be called a *property notation*.

We will concentrate on one behavioral property logic in this paper, called RdTrLρ (RdTrL for Ready Trace Logic, and ρ for real time). This logic provides primitive properties for traces. Using a convention for the implicit presence of a trace variable and a temporal variable, each formula of RdTrLρ can be interpreted as a set of traces (at the corresponding level of technical abstraction).

1.2. Trace Theory.

It may be useful to explain why in our view trace theory is a necessary tool in the systems design and analysis area. Trace theory complements process algebra based on bisimulation semantics by being more abstract and allowing a full exploitation of the expressive power of linear time temporal logic. In addition, trace theory allows a very flexible expression of system properties related to *fairness*. There is an extensive literature about system description in terms of their set of traces, e.g., [BRO92, VIN90, MEY85].

The main degree of freedom in the development of trace theories lies in the additional decoration of the traces with information on how a computation could have proceeded in alternative ways. In Eindhoven, starting with [REM83], a group of researchers has developed a form of trace theory using undecorated traces. Many theoretical results have been found (e.g., [SNE85, KAL86]) and significant applications concerning integrated circuit design and verification, as well as foundational advances in the concepts of selftimed and delay insensitive systems, have been obtained (e.g., [UDD86, EBE89, BER92]).

In [BRHR84, OLH83], two forms of decorated trace theory have been introduced: failure set semantics and ready set semantics. This work provided a basis for theoretical CSP, a language that has been a platform for many subsequent studies (e.g. [HOA85]). Quite related to this work is the refusal sets model of Phillips [PHI87] and work by [HDN84] on testing.

In [PNU85], the barbed wire model was proposed. This is a decorated trace theory in which, for each action, the set of actions that a process might have done alternatively is recorded. In [BABK87] a similar model was proposed under the heading of a *ready trace set* model. Both proposals originate in the observation that for certain system construction techniques (e.g. broadcasting, priority mechanism), the distinguishing power of Rem's trace theory or that of [BRHR84] is not sufficient.

This paper proposes a version of trace theory that uses the ready trace set model of [BABK87], but slightly modified in order to accommodate a precise account of fairness and liveness and to support a mixed term formalism that exploits linear time ready trace logic as a system construction primitive (Definition 4.4). Our contribution is a systematic development of a version of a trace theory for the syntax of ACPρI [BAB91]. In particular, we develop an appropriate property language compatible with ACPρI. No novelty lies in any of the semantic techniques, but our ready trace theory provides a selfcontained explanation of a language for process description, as well as a preferred semantic model.

2. Real Time Ready Traces.

Due to the lack of space, we present here only one version of the theory: *real time ready traces*. Two other versions, *untimed (symbolic)* and *discrete time ready traces* can be derived from it, see [BABB93]. Throughout the paper, let A be a set, whose elements are used as atomic actions, and let $\delta \notin A$.

$RT\rho(A)$, the collection of real time ready traces over A, consists of the functions $f: \mathbb{R}_{\geq 0} \to \{\surd, \delta, \Omega\} \cup \{(U, a) : U \subseteq A \cup \{wait\}, a \in U\}$, with the conditions:

- $f(t) = \surd \Rightarrow \forall s>t\ f(s) = \surd$ terminated trace
- $f(t) = \delta \Rightarrow \forall s>t\ f(s) = \delta$ deadlocked trace
- $f(t) = \Omega \Rightarrow \forall s>t\ f(s) = \Omega$ trace with undefined tail (see Example 4.3)
- $f(0) = (U, a) \Rightarrow a=wait$ no action at 0.

If $f(t) = (U, a)$, we put $f(t)_1 = U$, $f(t)_2 = a$. Otherwise, these notions are undefined. We call the (U, a) *ready pairs*. We call a ready trace α *terminating* if there is a t with $\alpha(t) = \surd$, *deadlocking* if there is a t with $\alpha(t) = \delta$, *eventually undefined* if there is a t with $\alpha(t) = \Omega$.

We call a ready trace set V *ready closed* if $\forall \beta \in V\ \forall r \in \mathbb{R}_{\geq 0}\ \forall U \subseteq A \cup \{wait\}\ \forall a,b \in U$:

$$[\beta(r) = (U, a) \Rightarrow \exists\, \beta' \in V\ \beta'(r) = (U, b) \wedge \forall s<r\ \beta'(s) = \beta(s)]$$

(i.e., if V contains a trace that shows that an action b is ready at time r, then V contains also a trace that deviates from the previous one exactly by taking the action b at time r).

We call a ready trace set V *time deterministic* if

$$\forall \alpha,\beta \in V\ \forall r \in \mathbb{R}_{\geq 0}\ (\forall s < r\ \alpha(s)_2 = \beta(s)_2 = wait) \Rightarrow (\alpha(r)_1 = \beta(r)_1 \vee \alpha(r) = \surd \vee \beta(r) = \surd)$$

(i.e., if V contains two traces that have only waited until r, then they have the same ready set at r (unless one of them terminated); thus they made no choice by just waiting.)

We call a ready trace α *sufficiently defined* if

$$\forall t \in \mathbb{R}_{\geq 0}\ (\alpha(t) = \Omega \Rightarrow \{t' < t \mid \alpha(t') = (U, a) \wedge a \neq wait\} \text{ is infinite})$$

(i.e., a trace can only be eventually undefined after an infinite number of actions). A ready trace set is sufficiently defined if all its traces are sufficiently defined.

We call a ready trace set V *right closed* if $\forall W \subseteq V\ \forall t_1\ \forall t_2 \geq t_1$ (in particular $t_2 = \infty$)

$$\begin{aligned}
&(\ (\forall \alpha,\beta \in W\ (\forall t < t_1 \quad \alpha(t) = \beta(t) \wedge \\
&\qquad \forall t \in [t_1,t_2)\ ((\forall s \in [t_1,t)\ \alpha(s)_2 = \beta(s)_2 = wait) \Rightarrow \alpha(t)_1 = \beta(t)_1)) \wedge \\
&\quad \forall t \in [t_1,t_2)\ \exists \alpha \in W\ \forall s \in [t_1,t)\ \alpha(s)_2 = wait) \\
&\Rightarrow (\exists\, \alpha \in V\ (\forall t \in [t_1,t_2)\ \alpha(t)_2 = wait \wedge \\
&\qquad \forall \beta \in W \quad (\forall t < t_1\ \alpha(t) = \beta(t) \wedge \\
&\qquad\qquad \forall t \in [t_1,t_2)\ (\forall s \in [t_1,t)\ \beta(s)_2 = wait \Rightarrow \alpha(t)_1 = \beta(t)_1))))\ )
\end{aligned}$$

(i.e., if a process can wait until a time arbitrarily close to t_2, then it must contain a trace that can wait until (but not including) t_2. This also holds for a 'subprocess' of V: a subset W of traces of V that are the same until time t_1, after which they obey time-determinism). See Section 3 for further explanation. A *process ready trace set* is a ready trace set that is ready closed, time deterministic, sufficiently defined and right closed.

3. An Algebra of Process Ready Trace Sets.

Let $\gamma: A \cup \{\delta\} \times A \cup \{\delta\} \to A \cup \{\delta\}$ be a commutative and associative function with $\gamma(\delta, a) = \delta$ for all $a \in A \cup \{\delta\}$. γ is called a *communication function*, and $\gamma(a,b)$ represents the action that results if a and b occur simultaneously. Let PRTS be the class of process ready trace sets. As we only deal with the behavior of processes (i.e., their ready trace set) in this paper, we will usually call a process ready trace set a *process*.

We define the following operators on PRTS:

• U: PRTS $\to \mathbb{R}_{\geq 0} \cup \{\infty\}$	ultimate delay; ∞ denotes infinite delay: $\sup(\mathbb{R}_{\geq 0})$.
• $a(t), \delta(t) \in$ PRTS	atomic action and deadlock; $(a \in A, t \in \mathbb{R}_{\geq 0} \cup \{\infty\})$.
• $+, \cdot, \parallel$: PRTS $\times$ PRTS $\to$ PRTS	alternative, sequential and parallel composition.
• ∂_H: PRTS $\to$ PRTS	encapsulation. $(H \subseteq A)$.
• $\int_V$: $(V \to$ PRTS$) \to$ PRTS	$(V \subseteq \mathbb{R}_{\geq 0}$; for $\int_V \lambda v.x(v)$, we write $\int_{v \in V} x(v))$: the infinite alternative composition of $x(v)$, $v \in V$.

These operators have the following interpretation [.] (where a,b range over $A \cup \{wait\}$).

- $U(x) = \sup\{t \in \mathbb{R}_{\geq 0} : \exists \alpha \in [x]\ \forall t' < t\ \alpha(t')_2 = wait\}$
- $[\delta(t)] = \{\alpha\}$, where $\alpha(s) =$ $(\{wait\}, wait)$ for $s < t$, δ for $s \geq t$.
- $[a(t)] = \{\alpha\}$, where $\alpha(s) =$ $(\{wait\}, wait)$ for $s < t$, $(\{a\}, a)$ for $s = t$ and $\surd$ for $s > t$.
  In particular, $[\delta(\infty)] = [a(\infty)] = \{\alpha\}$, where for all s: $\alpha(s) = (\{wait\}, wait)$.

The interpretations of $x + y$, $x \cdot y$, etc. are obtained from [x] and [y] in two steps. First, there is a *selection* of traces from [x] and [y], in which traces from one process that are incompatible with the other process are discarded. Second, there is a *combination* of traces. This combination usually involves, for each point in time, the choice of an action from [x] or one from [y], and a combination of ready sets.

- $[x + y] = \{\alpha' : \alpha \in [x] \cup [y]$ and $U(\{\alpha\}) < U([x] \cup [y]) \Rightarrow \alpha(U(\{\alpha\})) \neq \delta\}$ *(discard a trace that ends in deadlock after only waiting, if the process allows a longer wait)*, where

$$\alpha'(r) = \begin{cases} (Z, a) & \text{if } \alpha(r)_2 = a,\ U(\{\alpha\}) = r, \text{ and } Z = \cup\{\beta(r)_1 : \beta \in [x] \cup [y] \text{ and } U(\{\beta\}) \leq r\} \\ & \textit{(if } \alpha \textit{ has only waited so far, then add readies from the other process)}, \\ \alpha(r) & \text{in all other cases.} \end{cases}$$

- $[x \cdot y] = \{\alpha \cdot_{rt} \beta : \alpha \in [x], \beta \in [y]$ and $\forall r\ (\alpha(r) \neq \surd \wedge wait \in (\beta(r)_1) \Rightarrow \beta(r)_2 = wait\}$
  (discard those traces from [y] *that start doing actions too early)*, where

$$(\alpha \cdot_{rt} \beta)(r) = \begin{cases} \alpha(r) & \text{if } \alpha(r) \neq \surd, \\ \beta(r) & \text{if } \alpha(r) = \surd \text{ and } \forall t < r\ (\alpha(t) = \surd \vee \beta(t)_2 = wait), \\ \delta & \text{if } \alpha(r) = \surd \text{ and } \exists t < r\ (\alpha(t) \neq \surd \wedge \beta(t)_2 \neq wait). \end{cases}$$

(β had to start before α was finished, which results in deadlock).

- $[x \parallel y] = \{\alpha \parallel_{rt} \beta : \alpha \in [x], \beta \in [y]$ and
  $\forall r\ ((\alpha(r) = (U, a) \wedge \beta(r) = (V, b) \wedge a,b \in A \wedge \gamma(a,b) = \delta) \Rightarrow$
  $(wait \notin U \cup V \wedge \forall c \in U\ \forall d \in V\ \gamma(c,d) = \delta))\}$, *(discard pairs of traces that deadlock because they fail to communicate, if one of the processes could wait, or if communication was possible)*, where

$$(\alpha \parallel_{rt} \beta)(r) = \begin{cases} \delta & \text{if } \alpha(r) = \delta \text{ or } \beta(r) = \delta \text{ or } \textit{(communication failed or fails:)} \\ & \exists t \leq r\ \alpha(t) = (U, a),\ \beta(t) = (V, b),\ a,b \in A,\ \gamma(a,b) = \delta, \\ \Omega & \text{if } \alpha(r) = \Omega \text{ or } \beta(r) = \Omega, \\ \beta(r) & \text{if } \alpha(r) = \surd, \\ \alpha(r) & \text{if } \beta(r) = \surd, \\ (Z, a) & \text{if } \alpha(r) = (U, a),\ \beta(r) = (V, wait), \\ (Z, b) & \text{if } \alpha(r) = (U, wait),\ \beta(r) = (V, b), \\ (Z, \gamma(a,b)) & \text{if } \alpha(r) = (U, a),\ \beta(r) = (V, b),\ a,b \in A,\ \gamma(a,b) \neq \delta. \end{cases}$$

  In the last three cases, $Z = U \cup V \cup \{\gamma(a,b) : a \in U,\ b \in V,\ \gamma(a,b) \neq \delta\}$.

- $[\partial_H(x)] = \{\partial_{H,rt}(\alpha) : \alpha \in [x]$ and $\forall r\ (\alpha(r) = (U, a) \wedge a \in H \Rightarrow U \subseteq H)\}$
  (discard traces showing an encapsulated action, where another action or waiting is possible), where

$$\partial_{H,rt}(\alpha)(r) = \begin{cases} \alpha(r) & \text{if } \alpha(r) \in \{\surd, \delta, \Omega\} \text{ and } \forall t < r\ (\alpha(t) = (U, a) \Rightarrow a \notin H) \\ (U\text{-}H, a) & \text{if } \alpha(r) = (U, a) \text{ and } \forall t \leq r\ (\alpha(t) = (U, a) \Rightarrow a \notin H) \\ \delta & \text{if } \exists t \leq r\ \alpha(t) = (U, a) \text{ and } a \in H. \end{cases}$$

• $[\int_{v\in V} x(v)] = \{\alpha' : \alpha \in \cup_{v\in V}[x(v)] \cup [\delta(\sup\{U(x(v)) : v \in V\}]$ *(see below)* and
$U(\{\alpha\}) < \sup\{U(x(v)) : v \in V\} \Rightarrow \alpha(U(\{\alpha\})) \neq \delta\}$ *(see* [x + y]), where

$\alpha'(r) =$ (Z, a) if $\alpha(r)_2 = a$, $U(\{\alpha\}) = r$, and $Z = \cup\{\beta(r)_1 : \beta \in \cup_{v\in V}[x(v)], U(\{\beta\}) \leq r\}$
$\alpha(r)$ in all other cases.

The reason for adding the trace $[\delta(\sup\{U(x(v)) : v \in V\}]$ when considering the integral is that otherwise the definitions would give us e.g. $a(1)\cdot\int_{1<t<2} b(t) + a(1)\cdot\int_{1<t<3} b(t) = a(1)\cdot\int_{1<t<3} b(t)$, which seems undesirable. Namely, all traces of $\int_{1<t<2} b(t)$ would have to do a b-step before time 2, thus no trace could signal that b is no longer ready at time 2. The addition of $\delta(2)$ solves this problem, and also makes the trace set of $\int_{1<t<2} b(t)$ right closed. On the other hand, adding $\delta(2)$ to $\int_{1<t\leq 2} b(t)$ makes no difference, because it already has a trace α with $\alpha(t)_2 = wait$ for all $t<2$ and $\alpha(2) = (\{b\},b)$.

In this way, we also avoid that $[\int_{t>0} \delta(t)] = \varnothing$, which would mean $\partial_{\{r,s\}}(\ (i(1)\cdot s(2) + j(1)) \parallel \int_{t>0} r(t)\ ) = i(1)\cdot c(2)$ in a context with $\gamma(r,s) = c$. (Compare with Section 6: it should not be possible that a component of a system, that decides internally whether or not to send a message, is forced to send one because another component is waiting to receive it.)

Claim 3.1. PTRS *with the above operators is a model of the process algebra* ACPρI.

Of course, more identities hold in this model, which is based on ready traces, than in ACPρI, which axiomatizes bisimulation.

4. Real Time Ready Trace Logic.

We proceed by giving the syntax and semantics of the primitives of a language describing real time ready trace sets, parametrised by the set of atomic actions A.

Definition 4.1. Let α be a real time ready trace.

α **sat** a(t) if $\alpha(t) = (U, a)$ for some $U \subseteq A\cup\{wait\}$
(In particular, if α **sat** wait(t), then the system does not perform an action at t.)

α **sat** R(a, t) if $\alpha(t) = (U, b)$ for some $U \subseteq A\cup\{wait\}$, $a,b \in U$
(In particular, α **sat** R(wait, t), then the system need not perform an action at t.)

α **sat** $\surd$(t) if $\alpha(t) = \surd$

α **sat** δ(t) if $\alpha(t) = \delta$

α **sat** Ω(t) if $\alpha(t) = \Omega$

The real time ready trace language RdTrLρ(A) consists of the primitives mentioned above, augmented with constants for all objects in its time domain $\mathbb{R}_{\geq 0}$, a total ordering $<$ and binary operators +, ·, $\dot{-}$, and / on it. Furthermore, we have the following constructors from standard predicate logic: $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$, and $\exists$ (quantification can be over the time domain, the set of atomic actions A, or parts thereof). The semantics of all these language constructs is the standard one. In this way we obtain an explicit time temporal logic. For similar logics see [KOY89a]. (Note that we have some freedom in choosing operators on the time domains: different choices result in different languages with different expressive powers, see Example 5.4).

Definition 4.2. Let V be a real time ready trace set and φ a closed RdTrLρ(A) formula.

- V **sat** φ iff for all $\alpha \in V$: α **sat** φ.

- RTρ(A, φ) = $\{\alpha \in RT\rho(A) \mid \alpha$ **sat** $\varphi\}$, the set of traces that satisfies φ.

Example 4.3. We present some examples of meaningful RdTrLρ(A) formulas. The following formulas (where a,b range over A $\cup$ {wait}) are satisfied by all ready trace sets.

$\forall$ s,t t < s $\rightarrow$ (δ(t) $\rightarrow$ δ(s) $\wedge$ $\surd$(t) $\rightarrow$ $\surd$(s) $\wedge$ Ω(t) $\rightarrow$ Ω(s)).

$\forall$ t a(t) $\rightarrow$ R(a, t).

$\forall$ t a(t) $\wedge$ b(t) $\rightarrow$ a = b.

$\forall$ t (δ(t) $\vee$ $\surd$(t) $\vee$ Ω(t)) $\rightarrow$ $\neg$R(a, t).

In the following, we use RdTrLρ(A) to formulate an interesting classification of traces. A trace α is a *bounded action frequency trace* if

$$\alpha \textbf{ sat } \neg\exists\, t \in \mathbb{R}_{\geq 0}\, [(\forall s < t\ \exists\, r \in (s,t)\ \exists a \in A\ a(r))].$$

(This formula says that there is no time point t, such that an infinite sequence of actions occurs, having t as the limit of their time points.) A trace α is a *Zeno trace* if

$$\alpha \textbf{ sat } \exists\, t \in \mathbb{R}_{\geq 0}\, [(\forall\, s < t\ \exists\, r \in (s,t)\ \exists a \in A\ a(r))] \wedge$$
$$\forall\, t \in \mathbb{R}_{\geq 0}\, [(\forall\, s < t\ \exists\, r \in (s,t)\ \exists a \in A\ a(r)) \rightarrow \Omega(t)].$$

(After the unique limit point t, the trace is undefined.) A trace α is a *supertask trace* if

$$\alpha \textbf{ sat } \exists\, t \in \mathbb{R}_{\geq 0}\, [(\forall\, s < t\ \exists\, r \in (s,t)\ \exists a \in A\ a(r)) \wedge \neg\Omega(t)].$$

(The trace continues after a limit point.)

Notice that this defines three disjoint sets of traces: each trace either has bounded action frequency, or is Zeno or supertask. A set of traces has bounded action frequency if all traces it contains have bounded action frequency (but see Section 6.3), and a set of traces is called non-supertask if all traces it contains either have bounded action frequency or are Zeno. Usually, we consider non-supertask trace sets only.

Koymans [KOY89a] states in Section 5.3 (page 73) that *syntactical abstractness* imposes the restriction to specify message passing systems solely in terms of their input and output actions. It turns out that the decision to restrict the specification langauge to a syntactically abstract one is both clarifying and mathematically rewarding. We adhere to Koymans' criterion of syntactical abstractness by restricting the languages RdTrL as to use a, R(a) (a $\in$ A $\cup$ {wait}), $\surd$, δ, and Ω only.

Message passing systems are a special kind of processes. Therefore some compromise with Koymans' requirement on syntactical abstractness is to be expected. We think that our extension of the admitted propositions with readies, termination, deadlock, undefinedness and error captures a meaningful version of the concept of syntactical abstractness for temporal description formalisms in our specific context.

In contrast to [KOY89a], we have included natural numbers, booleans and various common operators in RdTrL. This makes the language more complex than Koymans allows, but on the other hand provides it with a uniform logical complexity. We maintain that even these extensions are consistent with Koymans' request for syntactical abstractness, because the additional mechanisms are fully standard in mathematics.

It is not decidable whether RTρ(A, φ) is ready closed, given a formula φ in RdTrLρ(A) (see [BABB93]). The fact that such a basic aspect of RTρ(A,φ) is not guaranteed and 'even worse' not decidable, justifies that the language RdTrLρ(A) is called a *property language* rather than a process description language (or in programming language terminology: a

process notation). So we distinguish between *process notations* such as ACPρI(A) and *property notations* such as RdTrLρ(A). Semantically, both determine subsets of RTρ(A), the difference being that a process notation always denotes a ready closed, time deterministic and right closed trace set (i.e., a process).

It should be noted that in the untimed case, a property notation can often be found very close to the process notation. A typical property notation is $\tau_I(X) = Q$. This asserts of process X that after abstraction from steps in I (i.e. turning X-actions in I into silent ones) X becomes equal to process Q. In principle, this technique can be used in the real time case just as well. The drawback however is that known abstraction operators reduce process complexity much less than in the untimed case. This is due to the fact that timing information cannot easily be suppressed by means of an abstraction operator.

We conclude that in the real time case, a distinction between a property notation and a process notation is justified, if not unavoidable.

Definition 4.4. Having defined ready trace language, we can now define the function $\varphi \,\square : \mathrm{PRTS} \to \mathrm{PRTS}$, for $\varphi \in \mathrm{RdTrL}\rho(A)$: for $x \in \mathrm{PTRS}$, $\varphi \,\square\, x = \{\alpha \in x : \alpha \textbf{ sat } \varphi\}$, if this set is a process ready trace set (undefined otherwise).

We notice that $x \textbf{ sat } \varphi$ iff $\varphi \,\square\, x = x$. An operator similar to $\varphi \,\square\, x$ has been defined on the bisimulation model in Parrow's thesis [PAR85]. "$\varphi \,\square\, x$" imports the property language into the process notation. We can, conversely, add a special primitive to RdTrLρ(A), that imports process notation into it.

Definition 4.5. For an expression P in a process notation, the primitive formula cd(P) denotes the *complete description* of P, that is, a property that is satisfied by a trace α iff α is in the trace set of P.

It depends on the expressibility of the logic and of the process notation, whether for every process P, cd(P) can be expressed in the other primitives of the logic. (Even if this is the case, the complete description primitive may be useful as syntactic sugar.) We expect that in most cases, even for simple processes involving recursion, the complete description is not expressible in the other primitives (see Example 5.4). The results of [KOY89b] point in the same direction. However, our conjecture is that each recursion-free process expression in ACPρI(A) has a complete description in RdTrLρ(A).

Example 4.6. We provide complete descriptions of two process expressions; the intention is to exemplify the difference in nature between ACPρI(A) and RdTrLρ(A). First,

$$\begin{aligned}\mathrm{cd}(a(7)) = \;& \forall t{<}7\,(\mathrm{wait}(t) \wedge \forall b{\in}A\, \neg R(b,t)) \wedge \\ & a(7) \wedge \neg R(\mathrm{wait},7) \wedge \forall b{\in}A\,(R(b,7) \to b{=}a) \wedge \forall t{>}7\ \surd(t).\end{aligned}$$

Second, let $B = \int_{t>0} \sum_{d\in D} r(d)(t)\cdot s(d)(t+1)$. Then

$$\begin{aligned}\mathrm{cd}(B) = \exists t_0{>}0\, \exists d{\in}D\ & \forall t \le t_0\, \forall a{\in}A\,(R(a,t) \leftrightarrow \exists e{\in}D\, a{=}r(e)) \wedge R(\mathrm{wait},t_0) \wedge \\ & r(d)(t_0) \wedge s(d)(t_0+1) \wedge \forall t\,(t \neq t_0 \wedge t < t_0+1 \to \mathrm{wait}(t)) \wedge \\ & \forall t\, \forall a{\in}A\,(t_0 < t < t_0{+}1 \to \neg R(a,t)) \wedge \\ & \forall a{\in}A\,(R(a,t_0{+}1) \to a{=}s(d)) \wedge \neg R(\mathrm{wait},t_0{+}1) \wedge \\ & \forall t > t_0+1\ \surd(t) \\ \vee\ & \forall t > 0\,(\mathrm{wait}(t) \wedge \forall a{\in}A\,(R(a,t) \leftrightarrow \exists e{\in}D\, a{=}r(e))).\end{aligned}$$

We conclude that for several simple processes a complete description is fairly complex and as a consequence, uninformative. But our hope is, that a complete construction of cd(B) is not necessary for proving a statement like $cd(B) \rightarrow \forall d,t\ \ r(d)(t) \rightarrow s(d)(t+1)$. Even without such a proof system, our logic is valuable as a means to *express* such a statement.

5. Recursion.

Now we consider recursive equations in this framework. As an example, we take $X = \int_{t>0} a(t) \cdot X$. A first solution is the ready trace set determined by the formula:

$\varphi_1 \equiv \forall t>0\ (R(wait, t) \wedge \forall b \in A\ (R(b, t) \leftrightarrow b=a)) \wedge \neg \exists\, t \in \mathbb{R}_{\geq 0}\ [(\forall s < t\ \exists\, r \in (s,t)\ a(r))]$.

This formula determines a ready trace set that is a solution and consists only of bounded action frequency traces. But the ready trace set determined by the formula:

$$\varphi_2 \equiv \varphi_1 \wedge \exists t\ \forall r, s\ (t < r < s \wedge a(r) \wedge a(s) \wedge \forall v\ (r < v < s \rightarrow \neg a(v)) \rightarrow s = r + 1).$$

also denotes a process ready trace set. The formula says that after a certain time t, if more a-actions come, then they must be one time-unit apart. Putting an additional a-action in front does not change this property. Thus we find that this process also denotes a solution of the recursion equation. Many more solutions exist, some containing Zeno and supertask traces. Thus this equation comes nowhere near to having a unique solution.

These considerations change if only finite ready traces are considered (thus bringing the approach closer to that of timed CSP [RER88]). The choice to work with complete (usually infinite) traces rather than with incomplete (finite) traces is motivated by the marvellous expressive power concerning different forms of liveness and fairness that is obtained if systems are described by means of these complete traces. This expressive power seems to be mainly responsible for the success of the temporal logic approach to concurrency.

If we want that recursive equations nevertheless define a unique process, then there are at least two options. One option is to define a metric on ready traces, and to allow only closed sets of ready traces as solutions. This leads us to the field of so-called *topological process theory*, initiated with the work of De Bakker & Zucker [BAZ82]. Restricting the class of ready trace sets by topological means in an appropriate way leads to a domain in which guarded equations have unique solutions. In the real time case, the topological techniques become much more complex, unfortunately. To our taste, these techniques are not satisfactory, and we propose a different way to have recursion equations define a process.

Our proposal for the introduction of recursively defined processes in extensional ready trace theory is the following, using techniques similar to [BEK88]. Let $\langle X(t_1, ..., t_k) \mid E \rangle$ denote a mapping from $\mathbb{R}^k$ to PRTS for each guarded recursive specification E involving a process variable X with k real parameters. For particular real values $r_1, ..., r_k$ a process ready trace set

$$P(r_1, ..., r_k) = \langle X(t_1, ..., t_k) \mid E \rangle (r_1, ..., r_k) = \langle X(r_1, ..., r_k) \mid E \rangle$$

is obtained as follows:

i. Determine a real time transition system from E for each of its (parametrised) process variables, following [BAB91]. (For simplicity, we do not consider the $\varphi\,\Box$ operator in this section.)

ii. Determine the ready trace set of the transition system thus obtained for $X(r_1, ..., r_k)$. This ready trace set is $P(r_1, ..., r_k)$.

We briefly recall the definition of a transition system from [BAB91, Section 4.4]; our presentation here is somewhat simplified.

Definition 5.1. A *state* is a pair $\langle p, t\rangle$, where p is a closed process expression or the termination symbol $\surd$, and $t \in \mathbb{R}_{\geq 0}$. A *transition* is a triple (source, action, target), usually denoted as $\text{source} \xrightarrow{\text{action}} \text{target}$, where source and target are states and action $\in A \cup \{\text{wait}\}$. Intuitively, $\langle p, t\rangle \xrightarrow{a} \langle p', t'\rangle$ means that the process p, when the time has become t, can wait until time t', at which time it performs an a-step and turns into p'.

A *transition system* is a set TS of transitions satisfying for all $a \in A \cup \{\text{wait}\}$:

- $\langle s, t\rangle \xrightarrow{a} \langle s', t'\rangle \in TS \Rightarrow (t < t' \wedge s \neq \surd \wedge (a = \text{wait} \Rightarrow s = s'))$,
- $\langle s, t\rangle \xrightarrow{a} \langle s', t'\rangle \in TS \Leftrightarrow \forall t'' \in (t, t')\, \langle s, t\rangle \xrightarrow{\text{wait}} \langle s, t''\rangle \in TS \wedge \langle s, t''\rangle \xrightarrow{a} \langle s', t'\rangle \in TS$.

The rules for determining a transition system from a recursive specification, as given in [BAB91], ensure that these properties hold. A *path* through a transition system TS is a *countable* sequence $t_1, t_2, \ldots$ of transitions from TS such that for all $i > 0$:

- t_{i+1} exists if and only if the target of t_i is the source of one or more transitions in TS,
- if t_{i+1} exists, then the target of t_i equals the source of t_{i+1}.
- if, for some $t \in \mathbb{R}_{\geq 0}$, all transitions $t_i, t_{i+1}, \ldots$ have wait as their action and a time $t' < t$ in their target, then there exists an $n \geq i$ such that for all sources s of $t_n, t_{n+1}, \ldots$, TS contains no transition $s \xrightarrow{a} \langle s'', t\rangle$ $(a \in A \cup \{\text{wait}\})$.

The last clause of this definition prevents paths that fail to proceed without being 'trapped'. For example, if TS contains the transition $\langle a(1),0\rangle \xrightarrow{a} \langle\surd,1\rangle$ and all the transitions that come with this one, then $\langle a(1), 0\rangle \xrightarrow{\text{wait}} \langle a(1), 1/2\rangle \xrightarrow{\text{wait}} \langle a(1), 3/4\rangle \xrightarrow{\text{wait}} \langle a(1), 7/8\rangle \xrightarrow{\text{wait}} \ldots$ is not a valid path. In contrast, if TS contains $\{\langle s,0\rangle \xrightarrow{a} \langle\surd,t\rangle \mid 0 < t < 1\}$, but *not* $\langle s,0\rangle \xrightarrow{a} \langle\surd,1\rangle$, nor any other transition $\langle s,0\rangle \xrightarrow{b} \langle s',1\rangle$ $(b \in A \cup \{\text{wait}\})$, then this path is valid. (E.g., $s = \int_{0<t<1} a(t)$.)

Definition 5.2. Let σ be a path through a transition system TS. The *ready trace determined by* σ *in* TS, $RT(\sigma,TS)$, is for each time $t \in \mathbb{R}_{\geq 0}$ defined as $RT(\sigma,TS)(t) =$

$\surd$	if σ is finite and the target of its last transition is $\langle\surd, t'\rangle$, with $t' < t$;
δ	if σ is finite and the target of its last transition is $\langle s, t'\rangle$, with $t' < t$ and $s \neq \surd$ (closed time stop);
δ	if σ is infinite, $t' < t$ for all targets $\langle s, t'\rangle$ of transitions in σ, and finitely many transitions in σ have an action other than wait (open time stop);
Ω	if σ is infinite, $t' < t$ for all targets $\langle s, t'\rangle$ of transitions in σ, and infinitely many transitions in σ have an action other than wait;
(U, a)	for $a \in A \cup \{\text{wait}\}$, if there exist s', t', s such that $\langle s', t'\rangle \xrightarrow{a} \langle s, t\rangle \in \sigma$ and $U = \{b \in A \cup \{\text{wait}\} \mid \exists s''\, \langle s', t'\rangle \xrightarrow{b} \langle s'', t\rangle \in TS\}$;
(U, wait)	if there exist $a \in A \cup \{\text{wait}\}$, s', t', s'', t'' such that $\langle s', t'\rangle \xrightarrow{a} \langle s'', t''\rangle \in \sigma$, $t' < t < t''$ and $U = \{\text{wait}\} \cup \{b \in A \mid \exists s\, \langle s', t'\rangle \xrightarrow{b} \langle s'', t\rangle \in TS\}$.

Now let p be a closed process expression, and let TS be the transition system associated to p as defined in [BAB91]. Then the ready trace set p is $\{RT(\sigma,TS) \mid \sigma \text{ is a path through TS and the source of the first transition of } \sigma \text{ is } \langle p,0\rangle\}$.

Proposition 5.3. *Let* p *be a closed process expression without recursion. Then the ready trace set* p *defined above coincides with* [p]*, as defined in Section 3. (It is especially interesting to check this claim for integrals over right-open intervals.)*

Note that we had *no* intention that unique solutions are obtained. Rather, we have defined a uniform way to select some solution of the system E and to use that to evaluate $\langle X(t_1, ..., t_k) \mid E\rangle$. Other selection mechanisms can lead to other interpretations of this syntax. Also note that the definitions imply that the ready trace defined by a path in a transition system is not a supertask. A special construction is needed for obtaining supertasks; this construction is introduced in Definition 5.5.

Example 5.4. We provide complete descriptions of the three clocks of [BAB91].

The process $\langle C_1(1) \mid C_1(t) = tick(t)\cdot C_1(t+1)\rangle$ (a perfect clock) has the following complete description:

$$\forall n \in \mathbb{N}\ (n \neq 0 \rightarrow tick(n) \wedge \forall a \in A\ (R(a, n) \rightarrow a = tick)) \wedge$$
$$\forall t \geq 0\ (\forall n \in \mathbb{N}\ t \neq n \vee t = 0 \rightarrow wait(t) \wedge \forall a \in A \neg R(a, t)).$$

The process $\langle C_2(1) \mid C_2(t) = \int_{v\in[t-0.01,t+0.01]} tick(v)\cdot C_2(t+1)\rangle$ (a clock allowing some fluctuation of the ticks) has the following complete description:

$$\forall t < 0.99\ wait(t) \wedge \forall a \in A \neg R(a, t) \wedge$$
$$\forall n \in \mathbb{N}\ (n \neq 0 \rightarrow$$
$$\exists t \in [n-0.01, n+0.01]$$
$$(tick(t) \wedge (t \neq n+0.01 \leftrightarrow R(wait, t)) \wedge \forall a \in A\ (R(a, t) \rightarrow a = tick)) \wedge$$
$$\forall r \in [n-0.01, t)\ (wait(r) \wedge R(tick, r) \wedge \forall a \in A\ (R(a, r) \rightarrow a = tick)) \wedge$$
$$\forall r \in (t, n+0.99)\ (wait(r) \wedge \forall a \in A \neg R(a, r))).$$

Whether or not the process $\langle C_3(1) \mid C_3(t) = \int_{v\in[t-0.01,t+0.01]} tick(v)\cdot C_2(v+1)\rangle$ (a clock cumulating the errors) has a finite complete description, depends on the expressiveness of the logic. An infinite series of consecutive choices, each one depending on the previous choice, has to be made. We can suggest the following higher order description:

$$\exists t_0, t_1, t_2, t_3, ...\quad t_0 = 0 \wedge \forall t < 0.99\ wait(t) \wedge \forall a \in A \neg R(a, t) \wedge$$
$$\forall n \in \mathbb{N}\ (t_{n+1} \in [t_n+0.99, t_n+1.01] \wedge tick(t_{n+1}) \wedge (t_{n+1} \neq t_n+1.01 \leftrightarrow R(wait, t_{n+1})) \wedge$$
$$\forall a \in A\ (R(a, t_{n+1}) \rightarrow a = tick)) \wedge$$
$$\forall t \in [t_n+0.99, t_{n+1})\ (wait(t) \wedge R(tick, t) \wedge \forall a \in A\ (R(a, t) \rightarrow a = tick)) \wedge$$
$$\forall t \in (t_{n+1}, t_{n+1}+0.99)\ (wait(t) \wedge \forall a \in A \neg R(a, t))).$$

But higher order logic is not needed here: we can encode the sequence $t_0, t_1, t_2, ...$ by one real number r. If we find an encoding such that the corresponding function decode, satisfying $\forall n \in \mathbb{N}\ decode(r,n) = t_n$, is expressible in the arithmetic of the logic, then we can replace in the above description $\exists t_0, t_1, t_2, ...$ by $\exists r$, and each t_n by decode(r,n).

The normal task axiom (NTA) excludes supertasks:

$$NTA \equiv \forall t\ ((\forall s < t\ \exists r \in (s,t)\ \exists a \in A\ a(r)) \rightarrow \Omega(t)), \text{ or equivalently:}$$
$$\neg\exists t \in \mathbb{R}_{\geq 0}\ [(\forall s < t\ \exists r \in (s,t)\ \exists a \in A\ a(r)) \wedge \neg\Omega(t)].$$

Suppose one intends to allow traces that do not satisfy the NTA, i.e. supertask traces. This can be useful for the conceptual analysis of certain communication protocols (see Section 6). We provide an operator that introduces supertask processes (i.e., ready trace sets that may contain supertasks).

Definition 5.5. The operator $\surd_\Omega$ is defined on ready traces by: $\surd_\Omega(\alpha)(t) = \surd$ if $\alpha(t) = \Omega$, $\alpha(t)$ otherwise. On ready trace sets, $\surd_\Omega$ is defined by applying it to each element of the set. A supertask is then obtained, e.g., as follows:

$$P = \surd_\Omega(\langle X(1) \mid X(t) = a(t)\cdot X(1 + t/2)\rangle)\cdot a(3).$$

It should be noticed that this operator is meaningless on transition systems. Consequently, a semantic model for recursion equations involving this operator requires more sophicated techniques than the ones outlined above.

6. (Non-)Existence of Coordinated Attack Protocols.

In this section, we look at real time ready trace theory. The protocol we consider, is the so-called *Coordinated Attack Protocol:* via communication through unreliable media M_{12} and M_{34}, processes P and Q should synchronise on a certain action they each perform independently (or at least, the execution of the two actions should be close enough in time). For more information, see [HAM90]. See Fig. 1.

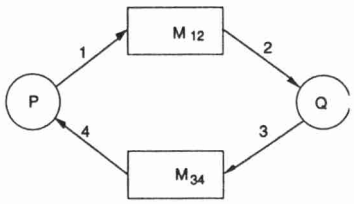

Fig. 1.

The story that goes with this picture is the following: P and Q are two generals that want to synchronise an attack on an army located between them, because only by working together they can beat this army. Their only means of communication is sending messengers that have to pass enemy lines. The messenger may arrive safely at the other army, or may be captured en route.

In the following, we use the convention, common in ACP, that s denotes a send action, r a receive action and i an internal action. Internal actions are subscripted for reference only. Send and receive actions are subscripted by a channel number; they communicate to a communication action c, also subscripted by the channel number. Thus the communication function γ satisfies: $\gamma(r_i(d),s_i(d)) = c_i(d)$ for $i = 1,2,3,4$, $d \in D$, In the interleaving framework of ACPρ, only one action at a time can occur. In order to allow simultaneous actions a and b, we make them 'communicate' to the so-called multi-action a&b. For simplicity, we shall not treat actions like a&b separately: we write X **sat** $a(t)\wedge b(t)$ instead of X **sat** (a&b)(t), violating the statement $\forall t\ a(t)\wedge b(t) \rightarrow a = b$ from Example 4.3.

6.1. The classical case.

We can describe the media as follows:

$$M_{12} = \int_{t>0} \sum_{d\in D} r_1(d)(t) \cdot (i_1(t+1)\cdot s_2(d)(t+2)+i_2(t+2)) \cdot M_{12}$$

$$M_{34} = \int_{t>0} \sum_{d\in D} r_3(d)(t) \cdot (i_3(t+1)\cdot s_4(d)(t+2)+i_4(t+2)) \cdot M_{34}$$

We may assume that the choice in the media is fair, in particular (and more specifically), let

$$\varphi_k = \forall t\, \exists s > t\; R(i_k, s) \;\rightarrow\; \forall t\, \exists s > t\; i_k(s),$$

then both media together can be written as

$$M = (\varphi_1 \,\square\, M_{12}) \,\|\, (\varphi_3 \,\square\, M_{34}).$$

In order to let the intended communication channels, *and only those*, function properly, we must

1. disallow all actions $r_1(d)$, $s_2(d)$, $r_3(d)$ and $s_4(d)$ by P and Q ($d \in D$),
2. encapsulate the system by the set $H = \{r_i(d), s_i(d) : i = 1,2,3,4,\ d \in D\}$,
3. define: $\gamma(r_i(d), s_i(d)) = c_i(d)$ for $i = 1,2,3,4,\ d \in D$,
   $\gamma(a,b) = a\&b$ otherwise.

Thus actions occurring at the same time, other than corresponding reads and sends, are completely independent.

The aim is that P and Q perform actions p_attack and q_attack at the same time: the final system must satisfy $\exists t$ p_attack&q_attack(t). This requirement, however, might be too strong for other reasons than the one we aim at (such as relativistic considerations); therefore we require that the attacks take place at most one time unit apart:

$$\Phi_{syn} \equiv \exists t_1, t_2\; (p\_attack(t_1) \wedge q\_attack(t_2) \wedge |t_1 - t_2| < 1).$$

Excluding one form of cheating, we require P **sat** $\forall t \neg$ q_attack(t). Excluding another one, we also require P **sat** $\forall s,t\ ((p\_attack(s) \wedge p\_attack(t)) \rightarrow s=t)$. Similarly for Q.

Due to our way of modeling parallelism, all components of a system share a global clock. We need to incorporate some mechanism that excludes using this clock for achieving synchronization. For example, the solution P = p_attack(1), Q = q_attack(1) must be avoided. What we want to express is that $U(P) = \infty$, that is

$$\forall t>0\ \exists \alpha \in V_P\ \forall t'<t\ \alpha(t')_2 = \text{wait}$$

(where V_P denotes the ready trace set of P), but this statement cannot be rephrased as P **sat** φ for any formula φ, because here the quantification over traces is *existential*, whereas the definition of P **sat** φ employs *universal* quantification. So let us say that the generals must first arrive at their positions, and that they cannot tell in advance how long this will take. Until (and included) their time of arrival, messages sent to them are lost:

$$P\_init(t) = \int_{0<u<t} \Big(\sum_{d \in D} r_4(d)(u) \cdot P\_init(t)\Big) + \sum_{d \in D} (r_4(d)\&p\_arrive)(t) + p\_arrive(t)$$

$$Q\_init(t) = \int_{0<u<t} \Big(\sum_{d \in D} r_2(d)(u) \cdot Q\_init(t)\Big) + \sum_{d \in D} (r_2(d)\&q\_arrive)(t) + q\_arrive(t).$$

Now we can write the whole system as:

$$R = \partial_H\Big(\int_{t>0} (P\_init(t) \cdot P(t)) \,\|\, M \,\|\, \int_{t>0} (Q\_init(t) \cdot Q(t))\Big).$$

This formulation works only under the additional assumption that for all t, the ready trace set of P(t) is non-empty, and the same for Q.

Theorem 6.1. *For all processes* P *and* Q *satisfying the requirements above,*

$$R \textbf{ sat } \Phi_{syn} \textit{ is false.}$$

PROOF. See [BABB93].

6.2. Supertasks.

We see that we cannot obtain global synchronisation, even if the processes can have arbitrary form. Now we show that, if we relax the definition of the channel by allowing infinitely many inputs in a finite amount of time, we can obtain global synchronisation. Consider the following channels:

$$M^*_{12} = \int_{t>0} \sum_{d\in D} r_1(d)(t)\cdot(M^*_{12} \parallel (i_1(t+1)\cdot s_2(d)(t+2) + i_2(t+2)))$$

$$M^*_{34} = \int_{t>0} \sum_{d\in D} r_3(d)(t)\cdot(M^*_{34} \parallel (i_3(t+1)\cdot s_4(d)(t+2) + i_4(t+2))) \quad .$$

Immediately after receiving an input, a new input can be received. Still, each attempted communication, whether successful or not, takes two time units. We redefine M (and thus R) by:

$$M = (\varphi_1 \,\square\, M^*_{12}) \parallel (\varphi_3 \,\square\, M^*_{34}).$$

We can get global synchronisation, if we allow supertasks (see Example 4.3 and Definition 5.5). Before we define P and Q, we define two auxiliary processes. The first process, $Send_i(t)$, is a supertask: it sends within one second, beginning at t, an infinite number of messages through channel i; then it terminates. We do not need any content in the messages: $D = \{d\}$.

$$Send_i(t) = \sqrt{}_\Omega(\langle S_i(t,1) \mid S_i(u,n) = s_i(d)(u)\cdot S_i(u+2^{-n},n+1)\rangle).$$

Channel fairness ensures that from such a burst of messages, at least one arrives. The second process, $Deaf_i$, simply accepts and ignores messages coming in through channel i.

$$Deaf_i = \int_{t>0} r_i(d)(t)\cdot Deaf_i.$$

The roles of P and Q in this protocol are asymmetric. After P arrives, it sends a burst of messages every second, while listening for an answer. Half a time unit after the first answer comes (meaning that Q has arrived), P attacks. When Q arrives, it waits until it receives a message from P. It answers this message by a one-second burst. If this burst starts at time s, it occupies the interval [s,s+1). Thus at least one message of the burst arrives at P in the interval [s+2,s+3), and P attacks in the interval [s+2.5,s+3.5). So it is safe for Q to attack at s+3. P and Q accept and ignore all incoming messages after the first one.

$$P(t) = \langle X(t+1) \mid X(u) = Send_1(u)\cdot X(u+1)\rangle \parallel \int_{u>t} (r_4(d)(u)\cdot(Deaf_4 \parallel p\_attack(u+0.5))).$$

$$Q(t) = \int_{u>t} (r_2(d)(u)\cdot(Deaf_2 \parallel (Send_3(u+1)\cdot q\_attack(u+4)))).$$

Theorem 6.2. *If we do* not *allow supertasks, then for all processes* P *and* Q *satisfying the requirements of Section 6.1 (recall that* R *is now defined using* M^*_{12} *and* M^*_{34}*),*

$$R \textbf{ sat } \Phi_{syn} \textit{ is false.}$$

PROOF. See [BABB93].

6.3 Variable Transmission Speed.

Instead of allowing infinitely many inputs in a finite amount of time, we can also allow the transmission speed to be variable. In this case, we can even obtain global synchronisation

with processes without supertasks. Each trace of the system will be a bounded action frequency trace, but for every $\varepsilon>0$, there is a trace in the system's trace set for which the bound is smaller than ε. Therefore it is not implementable. This protocol was suggested by J.F. Groote [GRO92].

Consider the following channels with programmable transmission speed (the only message to be transmitted is the transmission speed, so we remove D):

$$M''_{12} = \int_{t>0} \int_{r\in(0,1)} r_1(r)(t)\cdot(i_1(t+r)\cdot s_2(r)(t+2r) + i_2(t+2r)) \quad \cdot M''_{12}$$

$$M''_{34} = \int_{t>0} \int_{r\in(0,1)} r_3(r)(t)\cdot(i_3(t+r)\cdot s_4(r)(t+2r) + i_4(t+r2)) \quad \cdot M''_{34}$$

M and R are redefined accordingly.

Again, P and Q have asymmetric definitions. This time every eight time units P sends a message in order to determine if both processes are alive. When Q receives such a message, it will time out after seven time units. In order to make sure P knows this, one succesful message exchange suffices. This exchange, initiated by Q, will be done faster and faster, in order to fit within a six units time frame.

In fact, only a message from Q to P is necessary, but P replies in order to stop Q from sending more messages (otherwise, Q would have to be a supertask, sending an infinite number of messages before attacking). Of course, such a reply can get lost, but the fact that Q can send an arbitrary number of messages, means that an arbitrarily large subset of these messages arrives at P, thus P replies arbitrarily many times, and at least one of these replies must reach P. Formally,

$$P(t) = s_3(1/2)(t+8)\cdot P(t+8) + \int_{v\le t+7} r_2(r)(v)\cdot s_3(r)(v+r)\cdot P^*(v+10r) \quad ,$$

$$P^*(u) = p\_attack(u+1/2) + \int_{v\le u} r_2(r)(v)\cdot s_3(r)(v+r)\cdot P^*(u) \quad .$$

$$Q(t) = \int_{v>t} r_4(1/2)(v)\cdot s_1(1/2)(v+1/2)\cdot Q^*(v+1/2,\ 1/2,\ 7+v) \quad ,$$

$$Q^*(t, r, u) = r_4(r)(t+5r)\cdot q\_attack(u)) + s_1(r/2)(t+6r)\cdot Q^*(t+6r,\ r/2,\ u).$$

(Here, t is the time of the previous message sent, r is the transmission speed of that message, and u is the time-out time.)

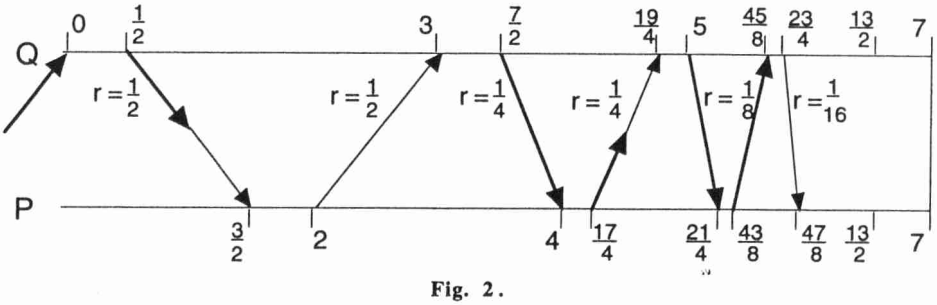

Fig. 2.

Fig. 2 depicts an example trace of the protocol, in which Q receives the first message from P at (relative) time 0 and both processes time out at time 7. The figure shows three attempted message exchanges, not counting the initial messages from P. The first attempted

exchange is interrupted in M''_{12} (the thin arrows denote messages that could have occurred, but did not occur, in this example). The second exchange is interrupted in M''_{34} and the third one succeeds. Therefore the fourth and later possible exchanges do not occur. P only replies if the corresponding message from Q arrives. As soon as a pair of corresponding messages succeeds, Q stops sending, and, as a consequence, so does P.

7. Conclusions, Problems, and Future work.

Several problems remain to be solved. We can divide them into two groups: problems concerning the algebra of process ready trace sets and problems concerning the logic.

Ready trace theory has appeared to be more complex than we expected. As a result, Section 3 has grown into a complex definition, of which one would like to prove that it corresponds to the intuitions behind it (for example by verifying Proposition 5.3).

The decision to add a summand $\delta(2)$ to $\int_{1<t<2} b(t)$ seems counterintuitive. But on the other hand it is plausible that $[\partial_{\{b\}}(a(1)\cdot\int_{1<t<2} b(t))] = [a(1)\cdot\delta(2)]$ (rather than $\emptyset$) and also that $a(1)\cdot\int_{1<t<3} b(t)$ differs from $a(1)\cdot\int_{1<t<2} b(t) + a(1)\cdot\int_{1<t<3} b(t)$. Adding a summand $b(2)$ could be an alternative, but that would be at least as counterintuitive, and technically even more complicated. That $\int_{t>0} a(t)$ has an ever waiting trace $\delta(\infty) = a(\infty)$ seems also reasonable.

As it is defined here, Ready Trace Logic can only be used to describe properties of *all* traces of a process. In Section 6.1, we wanted to state that a process has a certain trace (for every $t>0$). Thus the usefulness of RdTrL could be improved through replacing the implicit universal quantification over the trace variable by arbitrary quantification.

In order to be useful not only for specification, but also for verification, the operators on processes should be translated to 'connectives' in the logic: if P **sat** φ and Q **sat** ψ then P+Q **sat** $\varphi+\psi$. Notice that the laws for these connectives will differ from the equations of the process algebra, for example: $\varphi+\varphi \neq \varphi$. Defining these connectives in terms of RdTrL is not easy. It is probably worthwhile to study a language of which the primitives are on a higher level.

Apart from facilitating the translation from operators to connectives, this language could also solve another problem of RdTrL, namely that its notation is very explicit. When used on a high level of specification, this is an advantage of RdTrL, but on lower levels it becomes cumbersome to write down not only which actions are ready and take place, but also which actions are not ready, and when the process waits. Perhaps even a non-monotonic logic with a construction for invoking a Closed World Assumption [REI78] could be used (e.g., '$\neg$ R(a,t) holds, unless R(a,t) is explicitly stated.'). But a good understanding of RdTrL is obviously crucial, before this higher level logic can be defined.

Acknowledgements.

We thank J.F. Groote (Utrecht University) for discussions and helpful suggestions on Section 6. We thank J. Katoen and R. Koymans (Philips Research) for suggesting examples from industry.

References.

[BAB91] J.C.M. BAETEN & J.A. BERGSTRA, *Real time process algebra,* Formal Aspects of Computing 3 (2), 1991, pp. 142-188.

[BABB93] J.C.M. BAETEN, J.A. BERGSTRA & R.N. BOL, *A real time process logic*, report CSN 93/15, Dept. of Computing Science, Eindhoven University of Technology, 1993.

[BABK87] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Ready trace semantics for concrete process algebra with priority operator,* British Computer Journal 30 (6), 1987, pp. 498-506.

[BAZ82] J.W. DE BAKKER & J.I. ZUCKER, *Processes and the denotational semantics of concurrency,* I&C 54, 1982, pp. 70-120.

[BEK84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication,* Inf. & Control 60, 1984, pp. 109-137.

[BEK88] J.A. BERGSTRA & J.W. KLOP, *A complete inference system for regular processes with silent moves,* in: Proc. Logic Coll. 1986, Hull (F.R. Drake & J.K. Truss, eds.), North-Holland 1988, pp. 21-81.

[BER92] C.H. VAN BERKEL, *Handshake circuits: an intermediary between communicating processes and VLSI,* Ph.D. Thesis, Eindhoven University of Technology 1992.

[BRHR84] S.D. BROOKES, C.A.R. HOARE & A.W. ROSCOE, *A theory of communicating sequential processes,* J. ACM 31 (3), 1984, pp. 560-599.

[BRO92] M. BROY, *Functional specification of time sensitive communication systems,* NATO ASI series, series F: computer and systems sciences, Vol. 88, pp. 325-367.

[EBE89] J.C. EBERGEN, *Translating programs into delay-insensitive circuits,* Tract 56, CWI Amsterdam 1989.

[VGL90] R.J. VAN GLABBEEK, *The linear time – branching time spectrum,* in: Proc. CONCUR'90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), Springer LNCS 458, 1990, pp. 278-297.

[GRO92] J.F. GROOTE, Personal communication, 1992.

[HAM90] J.Y. HALPERN & Y.O. MOSES, *Knowledge and common knowledge in a distributed environment,* J. ACM 37, 1990, pp. 549-587.

[HDN84] M. HENNESSY & R. DE NICOLA, *Testing equivalences for processes,* TCS 34, 1984, pp. 83-134.

[HOA85] C.A.R. HOARE, *Communicating sequential processes,* Prentice Hall 1985.

[KAL86] A. KALDEWAIJ, *A formalism for concurrent processes,* Ph.D. Thesis, Eindhoven University of Technology 1986.

[KOY89a] R.L.C. KOYMANS, *Specifying message passing and time-critical systems with temporal logic,* Ph.D. Thesis, Eindhoven University of Technology 1989.

[KOY89b] R.L.C. KOYMANS, *Specifying message passing systems requires extending temporal logic,* in: Proc. Temporal Logic in Specification (B. Banieqbal, H. Barringer & A. Pnueli, eds.), Springer LNCS 398, 1989, pp. 213-223.

[MEY85] J.-J. CH. MEYER, *Merging regular processes by means of fixed point theory,* TCS 45, 1985, pp. 193-260.

[OLD91] E.-R. OLDEROG, *Nets, Terms and Formulas*, Cambridge Tracts in Theor. Comp. Sci. 23, Cambridge University Press 1991.

[OLH83] E.-R. OLDEROG & C.A.R. HOARE, *Specification-oriented semantics for communicating processes,* in: Proc. ICALP 83 (J. Díaz, ed.), Springer LNCS 154, 1983, pp. 561-572.

[PAR85] J. PARROW, *Fairness properties in process algebra - with applications in communication protocol verification,* Ph.D. Thesis, Uppsala University 1985.

[PHI87] I.C.C. PHILIPS, *Refusal testing,* TCS 50, 1987, pp. 241-284.

[PNU85] A. PNUELI, *Linear and branching structures in the semantics and logics of reactive systems,* in: Proc. ICALP 85 (W. Brauer, ed.), Springer LNCS 194, 1985, pp. 15-32.

[REI78] R. REITER, *On closed world databases*, in: Logic and Databases (H. Gallaire and J. Minker, eds.), Plenum Press, 1978.

[RER88] G.M. REED & A.W. ROSCOE, *A timed model for communicating sequential processes,* TCS 58, 1988, pp. 249-261.

[REM83] M. REM, *Partially ordered computations, with applications to VLSI design,* in: Proc. Found. of Comp. Sci. IV.2 (J.W. de Bakker J. van Leeuwen, eds.), MC Tract 159, Math. Centre, Amsterdam 1983, pp. 1-44.

[SNE85] J.L.A. VAN DE SNEPSCHEUT, *Trace theory and VLSI design,* Springer LNCS 200, 1985.

[UDD86] J.T. UDDING, *Classification and composition of delay-insensitive circuits,* Ph.D. Thesis, Eindhoven University of Technology 1986.

[VIN90] E.P. DE VINK, *Designing stream based semantics for uniform concurrency and logic programming,* Ph.D. Thesis, Free University, Amsterdam 1990.