A Real Time Process Logic

J.C.M. Baeten
Department of Computing Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

J.A. Bergstra
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

R.N. Bol
Department of Computing Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

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 symbolic (untimed) and 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 on two examples: expressing the correctness of a concurrent alternating bit protocol, and demonstrating the (well-known) non-existence of so-called coordinated attack protocols. A connection is made with the metric temporal logic of Koymans.

1980 Mathematics Subject Classification (1985 revision): 68Q55, 68Q45, 68Q10.
1987 CR Categories: F.1.2, F.3.1, D.1.3, D.3.1.
Key words & Phrases: ready trace, process algebra, real time, process logic, coordinated attack protocol.
Note: The first two authors received partial support from ESPRIT Basic Research Action 7166, CONCUR2. The second author also received partial support from ESPRIT Basic Research Action 6454, CONFER.

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 *ready traces* (see Section 2.1). We define symbolic (untimed) and real time ready traces. 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 RTL (Ready Trace Language). Its syntax and semantics are closely related to ready trace theory. We compare the expressivity of RTL with the metric temporal logic of KOYMANS [KOY89a,b]. Two large examples are presented to show how RTL can be used. In the first example, we state the correctness of a concurrent alternating bit protocol. Our hope is, that RTL can also be helpful in proving such a correctness claim. The second example demonstrates how the logic can help in proving the non-existence of so-called coordinated attack protocols (which was proved on the ground of epistemic considerations in [HAM90]). We conclude by listing some shortcomings of the logics, thus pointing out topics for further research.

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.

1. CONCEPTUAL ABSTRACTION.

We need some explicit philosophical considerations in order to develop a robust terminology that will survive the embedding in a broader context. We cannot possibly provide a general ontology of objects and mechanisms that are used and/or needed in a theory of programming languages, systems and methods. So only a fragment of the relevant notions will be mentioned. Unavoidably, besides being incomplete, our discussion will be simplified and therefore imprecise.

1.1 LEVELS OF 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 directly 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).

1.2 AUTOMATON.

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 e.g. with and without roots, termination states, deadlocks, time stamps, probability assignments on transitions, fairness conditions on paths etc. A format that allows one to present automata is often called a program notation. Operational semantics will assign an automaton to a program. Of course the program notation will reflect the technical level of abstraction at which the automata are to be treated.

1.3 PROCESS.

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 will (in general) have an operational semantics and allow 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.

1.4 BEHAVIOR AND BEHAVIORAL PROPERTY.

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 to specify properties may be called a property notation.

We will concentrate on one behavioral property logic in this paper, called RTLρ (RTL for Ready Trace Language, 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 RTLρ can be interpreted as a set of traces (at the corresponding level of technical abstraction).

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*. The trace theory is equipped with several operators not present in ACP. We mention:

1. ${}_f\|_f$ fair merge (see Section 4.1.3);
2. ${}_f\|$ left fair merge;
3. $\|_f$ right fair merge;
4. $\varphi \,\square\, x$ export from x all traces that satisfy the temporal logic formula φ (see Section 5.9);

In Parrow's thesis [PAR85], an operator similar to $\varphi \,\square\, x$ has been defined on the bisimulation model. Probably, all operators mentioned above can be defined on some appropriate homomorphic pre-image of the bisimulation model. The fair merge operators disappear in the real time case, because they refer to the fairness of arbitrary interleaving, and real time interleaving is not arbitrary, but governed by time. Yet the $\varphi \,\square\, x$ operator can still be used to express fairness in this case.

There is a extensive literature about system description in terms of their set of traces. We mention a few treatments: [BRO92], [VIN90], [MEY85].

2.1 DECORATED TRACES.

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] and [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 (Section 5.9). Our contribution is a systematic development of a version of a trace theory for the syntax of ACP [BEK84] and its real time version of [BAB91]. In particular, we can develop appropriate property languages RTL (Ready Trace Logic) compatible with ACP. 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.

3. READY TRACE SETS.

We present two versions of the theory: *untimed (symbolic) ready traces* and *real time ready traces*. A third version, *discrete time ready traces* can be easily derived on the basis of the information we give. Throughout the paper, let A be a set, whose elements will be used as atomic actions, and let $\delta \notin A$.

3.1 SYMBOLIC READY TRACES.

The collection of symbolic ready traces over A, $RT_S(A)$, consists of the functions $f: \omega \to B$ (where ω is the ordinal of the natural numbers), with $B = \{\surd, \delta\} \cup \{(U, a) : U \subseteq A, a \in U\}$, that meet the following conditions:

- $f(n) = \surd \;\Rightarrow\; f(n+1) = \surd$ terminated trace
- $f(n) = \delta \;\Rightarrow\; f(n+1) = \delta$ deadlocked trace.

The (U, a) are called *ready pairs.* We put RT* for the set of all finite sequences of ready pairs. Concatenation $*$ is defined on finite and infinite sequences in the usual way.

We call a ready trace α *terminating* if there is n with $\alpha(n) = \surd$, *deadlocking* if there is n with $\alpha(n) = \delta$.

We call a ready trace set V *ready closed* if

$$\forall \sigma \in RT^*, U \subseteq A, a,b \in U, \beta \in RT_S(A)\, [\sigma*(U,a)*\beta \in V \;\Rightarrow\; \exists \beta' \in B^\omega\, (\sigma*(U,b)*\beta' \in V)]$$

(i.e., if V contains a trace that shows that an action b is ready after σ, then V contains also a trace that indeed takes the action b after σ).

We call a ready trace set V *time deterministic* if $\forall \alpha,\beta \in V\, [\alpha(0) = (U, a) \wedge \beta(0) = (U', a') \Rightarrow U = U']$ (i.e., all traces have the same ready set before their first action).

A *process ready trace set* is a ready trace set that is ready closed and time deterministic.

3.2 REAL TIME READY TRACES.

For the description of real time ready traces, we need one extra symbol Ω, denoting undefinedness (see 5.5.3). $RT\rho(A)$, the collection of real time ready traces over A, consists of the functions $f: \mathbb{R}_{\geq 0} \to B+$, with $B+ = \{\surd, \delta, \Omega\} \cup \{(U, a) : U \subseteq A\cup\{wait\}, a \in U\}$, 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
- $f(0) = (U, a) \Rightarrow a=wait$ 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 a ready trace α *terminating* if there is t with $\alpha(t) = \surd$, *deadlocking* if there is t with $\alpha(t) = \delta$, *eventually undefined* if there is 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 have not made any 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 also $t_2 = \infty$)

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

(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 4.2.2 for further explanation.

A *process ready trace set* is a ready trace set that is ready closed, time deterministic, sufficiently defined and right closed.

3.3 DISCRETE TIME READY TRACES.

In this case, the collection of discrete time ready traces consists of the functions $f: \omega \times \omega \to B+$, as in Section 3.2. $f(n, k)$ stands for action number k in time slice n. Definitions can be adapted from the previous cases.

4. 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*.

4.1 SYMBOLIC READY TRACE ALGEBRA.

4.1.1 OPERATORS.

In the symbolic case, we define the following operators on PRTS:

- $\tilde{a} \in PRTS$ for $a \in A$
- $\tilde{\delta}, 0 \in PRTS$
- $+, \cdot, \parallel : PRTS \times PRTS \to PRTS$
- $\partial_H: PRTS \to PRTS$ for $H \subseteq A$
- $\theta_{\leq}: PRTS \to PRTS$ for $\leq$ a partial ordering on A

Abusing notation, we will often write δ instead of $\tilde{\delta}$ and a instead of $\tilde{a}$.

4.1.2 INTERPRETATION.

The interpretation [.] of these operators is as follows:

- $[0] = \varnothing$.
- $[\tilde{\delta}] = \{\delta^{\omega}\}$.
- $[\tilde{a}] = \{(\{a\}, a) * \surd^{\omega}\}$.
- $[x + y] = \cup\{\alpha +_{rt} \beta : \alpha \in [x], \beta \in [y]\}$, where

$$\alpha +_{rt} \beta = \begin{cases} \{\beta\} & \text{if } \alpha(0) = \delta \\ \{\alpha\} & \text{if } \beta(0) = \delta \\ \{\alpha, \beta\} & \text{if } \alpha(0) = \surd \text{ or } \beta(0) = \surd \text{ and the other not } \delta \\ \{(U \cup V, a) * \alpha', (U \cup V, b) * \beta'\} & \text{if } \alpha = (U, a) * \alpha', \beta = (V, b) * \beta'\ . \end{cases}$$

- $[x \cdot y] = \{\alpha \cdot_{rt} \beta : \alpha \in [x], \beta \in [y]\}$, where

$$(\alpha \cdot_{rt} \beta)(n) = \begin{cases} \alpha(n) & \text{if } \alpha(n) \neq \surd \\ \beta(n - k) & \text{if } n \geq k, \alpha(k) = \surd, \forall m < k\ \alpha(m) \neq \surd. \end{cases}$$

• $[x \parallel y] = \{\alpha \parallel_{rt}^{g} \beta : \alpha \in [x], \beta \in [y], g: \omega \to \{0,1,2\}$† and
$\alpha(|\{k < n : g(k) \in \{0, 2\}\}|) = (U,a), \beta(|\{k < n : g(k) \in \{1, 2\}\}|) = (V,b), g(n)=2 \to \gamma(a,b)\neq\delta\}$,
where, putting $k_1 = |\{k < n : g(k) \in \{0, 2\}\}|$, $k_2 = |\{k < n : g(k) \in \{1, 2\}\}|$,

$$(\alpha \parallel_{rt}^{g} \beta)(n) = \begin{cases} \surd & \text{if } \alpha(k_1) = \beta(k_2) = \surd \\ \delta & \text{if } \alpha(k_1) = \delta, \beta(k_2) \in \{\delta,\surd\} \text{ or } \beta(k_2) = \delta, \alpha(k_1) \in \{\delta,\surd\} \\ (U, a) & \text{if } \alpha(k_1) = (U, a), \beta(k_2) \in \{\delta, \surd\} \\ (V, b) & \text{if } \alpha(k_1) \in \{\delta,\surd\}, \beta(k_2) = (V, b) \\ (Z, a) & \text{if } \alpha(k_1) = (U, a), \beta(k_2) = (V, b), g(n) = 0 \\ (Z, b) & \text{if } \alpha(k_1) = (U, a), \beta(k_2) = (V, b), g(n) = 1 \\ (Z, \gamma(a, b)) & \text{if } \alpha(k_1) = (U, a), \beta(k_2) = (V, b), g(n) = 2 \end{cases}$$

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

• $[\partial_H(x)] = \{\partial_{H,rt}(\alpha) : \alpha \in [x] \text{ and } \alpha(n) = (U, a) \wedge a \in H \to U \subseteq H\}$, where if for some $k < n$, $\partial_{H,rt}(\alpha)(k) = \delta$, then $\partial_{H,rt}(\alpha)(n) = \delta$. Otherwise, we have the following 3 cases:

$$\partial_{H,rt}(\alpha)(n) = \begin{cases} (U\text{-}H, a) & \text{if } \alpha(n) = (U, a), a \notin H \\ \surd & \text{if } \alpha(n) = \surd \\ \delta & \text{if } \alpha(n) = \delta \text{ or } \alpha(n) = (U, a), a \in H. \end{cases}$$

• $[\theta_{\leq}(x)] = \{\theta_{\leq,rt}(\alpha) : \alpha \in [x] \text{ and } \alpha(n) = (U, a) \to \neg\exists b \in U\ b > a\}$, where

$$\theta_{\leq,rt}(\alpha)(n) = \begin{cases} \alpha(n) & \text{if } \alpha(n) \in \{\surd, \delta\} \\ (\{u \in U : \neg\exists b \in U\ b > u\}, a) & \text{if } \alpha(n) = (U, a). \end{cases}$$

We can show that these definitions turn the set of process ready trace sets into a model for ACP, the Algebra of Communicating Processes of [BEK84], [BAW90]. The addition of the priority operator gives a model for ACP_θ, ACP with priorities of [BABK86], [BAW90]. The zero process was introduced in [BAB90]. In addition, we sometimes have occasion to use conditional operators, viz. $x \lhd \varphi \rhd y$ meaning *if* φ *then* x *else* y, and $\varphi :\to x$ meaning *if* φ *then* x (*else* δ). On trace sets, it is straightforward to define these operators.

4.1.3 FAIR MERGE.

Now we define the extra operators:

$$x\ {}_f\!\parallel_f y = \{\alpha \parallel_{rt}^{g} \beta : \alpha \in x, \beta \in y, \forall n\, \exists\, k,m > n\ g(k) \in \{0, 2\} \wedge g(m) \in \{1, 2\}\}$$

$$x\ {}_f\!\parallel y = \{\alpha \parallel_{rt}^{g} \beta : \alpha \in x, \beta \in y, \forall n\, \exists\, k > n\ g(k) \in \{0, 2\}\}$$

$$x \parallel_f y = \{\alpha \parallel_{rt}^{g} \beta : \alpha \in x, \beta \in y, \forall n\, \exists\, k > n\ g(k) \in \{1, 2\}\}.$$

Notice that $x\ {}_f\!\parallel_f y = x\ {}_f\!\parallel y \cap x \parallel_f y$.

---

† The function g describes a ‘scheduler’ for the processes x and y: if $g(n) = 0$, then only x can take a step; if $g(n) = 1$, then only y can take a step, and if $g(n) = 2$, then both can take a step, resulting in communication. If one of the processes is terminated or in deadlock, then we can always take $g(n) = 2$ (this seems counterintuitive, but it facilitates the definition of the fair merge).

4.2 REAL TIME READY TRACE ALGEBRA.

4.2.1 OPERATORS.

In the real time case, 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 for $a \in A$, $t \in \mathbb{R}_{\geq 0} \cup \{\infty\}$
- $0 \in$ PRTS
- $+, \cdot, \|$: PRTS $\times$ PRTS $\to$ PRTS
- ∂_H: PRTS $\to$ PRTS for $H \subseteq A$
- $\theta_{\leq}$: PRTS $\to$ PRTS for $\leq$ a partial ordering on $A \cup \{wait\}$
- $\int_V$: (V $\to$ PRTS) $\to$ PRTS for $V \subseteq \mathbb{R}_{\geq 0}$ (instead of $\int_V \lambda v.x(v)$, we write $\int_{v \in V} x(v)$).
- $\gg$: $\mathbb{R}_{\geq 0} \times$ PRTS $\to$ PRTS
- $\gg$: PRTS $\times \mathbb{R}_{\geq 0} \to$ PRTS

4.2.2 INTERPRETATION.

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

- $U(x) = \sup\{t \in \mathbb{R}_{\geq 0} : \exists \alpha \in [x]\ \forall t'<t\ \alpha(t')_2 = wait\}$
- $[0] = \emptyset$.
- $[\delta(t)] = \{\alpha\}$, 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 $s = t$; $\surd$ for $s > t$.

In particular, $[\delta(\infty)] = [a(\infty)] = \{\alpha\}$, where for all s: $\alpha(s) = (\{wait\}, wait)$.

- $[x + y] = \{\alpha' : \alpha \in [x] \cup [y]$ and $\neg\exists r\ (\exists \beta \in [x] \cup [y]: (\beta(r) \neq \delta \wedge \forall t < r\ \beta(t)_2 = wait) \wedge$ $\alpha(r) = \delta \wedge \forall t < r\ (\alpha(t) = \delta \vee \alpha(t)_2 = wait))\ \}$, where

$$\alpha'(r) = (U, a) \quad \text{if } \alpha(r)_2 = a,\ \forall s<r\ \alpha(s)_2 = wait \text{ and } U = \cup\{\beta(r)_1 : \beta \in [x] \cup [y] \text{ and } \forall s < r\ \beta(s)_2 = wait\}$$
$$\alpha'(r) = \alpha(r) \quad \text{in all other cases.}$$

- $[x \cdot y] = \{\alpha \cdot_{rt} \beta : \alpha \in [x], \beta \in [y]$ and $\forall r\ (\alpha(r) \neq \surd \wedge wait \in (\beta(r)_1) \to \beta(r)_2 = wait\}$, where

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

- $[x \| y] = \{\alpha \|_{rt} \beta : \alpha \in [x], \beta \in [y]$ and $\forall r\ (\alpha(r) = (U, a) \wedge \beta(r) = (V, b) \wedge a,b \in A \wedge \gamma(a,b) = \delta \to$ $wait \notin U \cup V \wedge \forall c \in U\ \forall d \in V\ \gamma(c,d) = \delta)\}$, where

$$(\alpha \|_{rt} \beta)(r) = \begin{cases} \delta & \text{if } \alpha(r) = \delta \text{ or } \beta(r) = \delta \text{ or } \exists t \leq r\ \alpha(t) = (U, a), \beta(t) = (V, b), a,b \in A, \gamma(a,b) = \delta \\ \Omega & \text{if } \alpha(r) = \Omega \text{ or } \beta(r) = \Omega \\ \beta(r) & \text{if } \alpha(r) = \surd \end{cases}$$

$\alpha(r)$	if $\beta(r) = \surd$
(Z, a)	if $\alpha(r) = (U, a)$, $\beta(r) = (V, wait)$
(Z, b)	if $\alpha(r) = (U, wait)$, $\beta(r) = (V, b)$
$(Z, \gamma(a,b))$	if $\alpha(r) = (U, a)$, $\beta(r) = (V, b)$, $a,b \in A$, $\gamma(a,b) \neq \delta$

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

- $[\partial_H(x)] = \{\partial_{H,rt}(\alpha) : \alpha \in [x]$ and $\forall r\ (\alpha(r) = (U, a) \wedge a \in H \to U \subseteq H)\}$, where

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

- $[\theta_{\leq}(x)] = \{\theta_{\leq,rt}(\alpha) : \alpha \in [x]$ and $\forall r\ (\alpha(r) = (U, a) \to \neg\exists b \in U\ b > a)\}$, where

$$\theta_{\leq,rt}(\alpha)(r) = \begin{cases} \alpha(r) & \text{if } \alpha(r) \in \{\surd, \delta, \Omega\} \\ (\{u \in U : \neg\exists b \in U\ b > u\}, a) & \text{if } \alpha(r) = (U, a). \end{cases}$$

- $[\int_{v\in V} x(v)] = \{\alpha' : \alpha \in \cup_{v\in V} [x(v)] \cup [\delta(\sup\{U(x(v)) : v \in V\}]$ and $\neg\exists r\ (\exists\beta \in \cup_{v\in V} [x(v)] : (\beta(r) \neq \delta \wedge \forall t < r\ \beta(t)_2 = wait) \wedge \alpha(r) = \delta \wedge \forall t < r\ (\alpha(t) = \delta \vee \alpha(t)_2 = wait))\}$, where

$$\alpha'(r) = (U, a) \quad \text{if } \alpha(r)_2 = a,\ \forall s<r\ \alpha(s)_2 = wait \text{ and } U = \cup\{\beta(r)_1 : \beta \in \cup_{v\in V} [x(v)] \text{ and } \forall s < r\ \beta(s)_2 = wait\}$$
$$\alpha'(r) = \alpha(r) \quad \text{in all other cases.}$$

- $[t \gg x] = \{t \gg \alpha : \alpha \in [x], \forall s \leq t\ (wait \in (\alpha(s))_1 \to (\alpha(s))_2 = wait)\}$, where

$$(t \gg \alpha)(r) = \begin{cases} (\{wait\}, wait) & \text{if } r < t \\ \delta & \text{if } r \geq t, \exists s \leq t\ (\alpha(s) = (U, a) \wedge wait \notin U) \vee \alpha(s) = \delta \\ \alpha(r) & \text{if } r \geq t, \forall s \leq t\ \alpha(s)_2 = wait. \end{cases}$$

- $[x \gg t] = \{\alpha \gg t : \alpha \in [x]\}$

$$(\alpha \gg t)(r) = \begin{cases} \alpha(r) & \text{if } r < t \\ \delta & \text{if } r \geq t, \forall s<t\ \alpha(s)_1 = \{wait\} \\ \alpha(r) & \text{if } r \geq t, \exists s<t\ \alpha(s) \in \{\delta, \Omega, \surd\} \text{ or } \alpha(s) = (U, a), a \neq wait \end{cases}$$

We can show that these definitions turn the set of process ready trace sets into a model for ACPρI(A), the Real Time Process Algebra with integration of [BAB91]. The addition of the priority operator was discussed in [BAB92b].

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 $\int_{t>0} \delta(t) = 0$, 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 8: 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 on the ground that another component is waiting to receive it.)

4.2.3 TIME STOPS, ZERO PROCESS.
In the light of the definitions above, we can reexamine the remarks on closed and open time stops in [BAB92b]. First notice that we have both kinds of time stops, i.e.

- open time stop $\delta(t)$ defined above
- closed time stop $\hat{\delta}(t)$ defined by

$\hat{\delta}(t)(s) = (\{wait\}, wait)$ for $s \leq t$
$\hat{\delta}(t)(s) = \delta$ for $s > t$.

In our setting, these are distinct processes. However, we get different results than in [BAB92b] as to how we can define closed time stops in the syntax. We find

$$\theta_{\leq}(\int_{t>1} a(t)) = 0 \qquad \text{if } wait \leq a,$$

so this does not give a closed time stop as surmised in [BAB92b]. On the other hand, the obvious generalisation of the definition of merge will give

$$\mathop{\|}_{r>t} \delta(r) = \hat{\delta}(t).$$

We also see that the zero process of [BAB90] can emerge by application of process operators. Therefore, we included it in our process syntax.

4.3 DISCRETE TIME READY TRACE ALGEBRA.
Along the same lines as above, we can define a discrete time variant. This will yield an algebra that is a model of the Discrete Time Process Algebra of [BAB92a].

5. REAL TIME READY TRACE LANGUAGE.

We proceed by giving the syntax and semantics of the primitives of a language describing symbolic, respectively real time ready trace sets. Both versions are parametrised by the set of atomic actions A. A discrete time version can be constructed similarly. We study the real time ready trace language in more detail.

5.1 DEFINITION (symbolic case). Let α be a symbolic ready trace.

α **sat** $a(n)$ $\alpha(n) = (U, a)$ for some $U \subseteq A$
α **sat** $R(a, n)$ $\alpha(n) = (U, b)$ for some $U \subseteq A$, $a,b \in U$
α **sat** $\surd(n)$ $\alpha(n) = \surd$
α **sat** $\delta(n)$ $\alpha(n) = \delta$.

5.2 DEFINITION (real time case). Let α be a real time ready trace.

α **sat** $a(t)$ — $\alpha(t) = (U, a)$ 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** $R(a, t)$ — $\alpha(t) = (U, b)$ 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** $\surd(t)$ — $\alpha(t) = \surd$

α **sat** $\delta(t)$ — $\alpha(t) = \delta$

α **sat** $\Omega(t)$ — $\alpha(t) = \Omega$

5.3 DEFINITION. We can now define symbolic and real time ready trace language, $RTL(A)$ and $RTL\rho(A)$ respectively. In the symbolic case, the time domain is $\mathbb{N}$, in the real time case, the time domain is $\mathbb{R}_{\geq 0}$. These languages have for their respective time domains: constants for all objects, a total ordering $<$ and binary operators $+$, $\cdot$, $\dot{-}$, and in the real time case $/$. Furthermore, we have the following constructors from standard predicate logic: $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$ (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. Other choices result in different languages with different expressive powers.

5.4 DEFINITIONS. Let V be a real time ready trace set and φ a closed $RTL\rho(A)$ formula.

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

$RTS\rho(A, \varphi) = \{\alpha \in RT\rho(A) \mid \alpha \text{ **sat** } \varphi\}$.

5.5 EXAMPLES. We present some examples of meaningful $RTL\rho(A)$ formulas.

5.5.1 SOME PROPERTIES THAT MIGHT REFLECT CONSTRAINTS FROM THE HARDWARE.

1. $\forall t\, \forall a \in A\;\; a(t) \rightarrow \exists s < t\, \forall u\, (s \leq u \leq t \;\rightarrow\; R(a, u))$.

This says that an action can only be performed if it has consistently been enabled during some time interval just preceding the time of the action.

2. $\forall t\, \forall d \in D\;\; r_i(d)(t) \;\rightarrow\; \exists s \in (t{+}c{-}\varepsilon, t{+}c{+}\varepsilon)\;\; s_j(d)(s)$.

This formula asserts that data read at i will be returned at j with a delay c with margin ε.

3. $\forall t\, \forall d \in D\;\; r_i(d)(t) \;\rightarrow\; \forall s \in (t, t{+}c)\; \forall e \in D\;\; \neg R(r_i(e), s)$.

This means that after an input at i a delay of at least c must be respected before a new input is taken.

5.5.2 AXIOMS.

The following formulas (where a,b range over $A \cup \{wait\}$) are satisfied by all ready trace sets. We might thus call them axioms (without suggesting that they play a special role in some proof system).

$\forall\, s,t\;\; t < s \;\rightarrow\; (\delta(t) \rightarrow \delta(s) \;\wedge\; \surd(t) \rightarrow \surd(s) \wedge \Omega(t) \rightarrow \Omega(s))$

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

$\forall\, t\;\; a(t) \wedge b(t) \;\rightarrow\; a = b$

$\forall t\ \delta(t) \to \neg R(a, t)$
$\forall t\ \surd(t) \to \neg R(a, t)$
$\forall t\ \Omega(t) \to \neg R(a, t)$.

5.5.3 A CLASSIFICATION OF TRACES.
A trace α is a *bounded action frequency trace* if α **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 α **sat** $\exists t \in \mathbb{R}_{\geq 0}\ [(\forall s < t\ \exists r \in (s,t)\ \exists a \in A\ a(r))] \wedge$
$\forall t \in \mathbb{R}_{\geq 0}\ [(\forall s < t\ \exists r \in (s,t)\ \exists a \in A\ a(r)) \to \Omega(t)]$.
(There is exactly one limit point t, after which the trace is undefined.)
A trace α is a *supertask trace* if α **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 8.6), 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.

5.6 DISCUSSION. [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 RTL as to use a, $R(a)$ (wait in case of real time), $\surd$, δ, Ω only.

Notice that 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.

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

5.7 PROPOSITION. It is not decidable whether $RTS\rho(A, \varphi)$ is ready closed.
PROOF: Consider the ready traces α, β given by:
- $\alpha(t) = (\{wait\}, wait)\ (t<1)$, $\alpha(1) = (\{a,b\}, a)$, $\alpha(t) = \delta\ (t>1)$;
- $\beta(t) = (\{wait\}, wait)\ (t<1)$, $\beta(1) = (\{a,b\}, b)$, $\beta(t) = \delta\ (t>1)$.

Notice that the ready trace set $\{\alpha, \beta\}$ is ready closed, but the ready trace set $\{\alpha\}$ is not. Now let $\varphi(x)$ be a RTL$\rho(A)$ formula with natural number variable x such that $\varphi(n)$ is not decidable. Moreover, choose formulas φ_α, φ_β such that RTS$\rho(A, \varphi_\alpha) = \{\alpha\}$, RTS$\rho(A, \varphi_\beta) = \{\beta\}$, e.g.
$\varphi_\alpha \equiv R(a, 1) \wedge R(b, 1) \wedge a(1) \wedge \neg b(1) \wedge \forall t{<}1\ \mathrm{wait}(t) \wedge \forall t{>}1\ \delta(t) \wedge$
$\forall t{<}1\ \forall c{\in}A\ \neg R(c, t) \wedge \forall c{\in}A\ (c{\neq}a \wedge c{\neq}b \rightarrow \neg R(c, 1))$.
Now we can define $\psi(x) \equiv \varphi_\alpha \vee (\varphi(x) \wedge \varphi_\beta)$, and we find that

$$\mathrm{RTS}\rho(A, \psi(n)) \text{ is ready closed } \Leftrightarrow \varphi(n) \text{ is true.}$$

Since the latter statement is undecidable, the former is too.

5.8 REMARK. The fact that basic aspects of RTS$\rho(A,\varphi)$, such as ready closure, are not guaranteed and 'even worse' not decidable, justifies that the language RTL$\rho(A)$ 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 RTL$\rho(A)$. Semantically, both determine subsets of RTS$\rho(A)$, the difference being that a process notation will always denote 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.

5.9 DEFINITION. Having defined ready trace language, we can now define the function
$\varphi \,\square : \mathrm{PRTS} \rightarrow \mathrm{PRTS}$, for $\varphi \in$ RTL(A) (symbolic or real time):
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 **sat** φ iff $\varphi \,\square\, x = x$.

5.10. COMPLETE DESCRIPTION PRIMITIVE.
The construction $\varphi \,\square\, x$ imports the property language into the process notation. We can, conversely, add a special primitive to RTL(A) and RTL$\rho(A)$, that imports process notation into it.

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, see e.g. Example 5.12.4.) We expect that in most cases, even for simple processes involving recursion, the complete description is not expressible in the other primitives (see Sections 6.5 and 6.6). The results of [KOY89b] point in the same direction.

5.11 CONJECTURE. Let p be a recursion-free process expression in ACPρI(A). Then p has a complete description φ in RTLρ(A).

5.12 EXAMPLES. We provide complete descriptions of some process expressions; the intention is to exemplify the difference in nature between ACPρI(A) and RTLρ(A).

1. $cd(0) = \text{false}$.
2. $cd(a(7)) =$
$\forall t<7\ (wait(t) \wedge \forall b\in A\ \neg R(b, t)) \wedge a(7) \wedge \neg R(wait, 7) \wedge \forall b\in A\ (R(b, 7) \rightarrow b=a) \wedge \forall t>7\ \surd(t)$.
3. Let $B = \int_{t>0} \sum_{d\in D} r_1(d)(t)\cdot s_2(d)(t + 1)$.

$$\begin{aligned}cd(B) = {} & \exists t_0>0\ \exists d_0\in D\ \ r_1(d_0)(t_0) \wedge s_2(d_0)(t_0 + 1) \wedge \\ & \quad \forall t\ (t\neq t_0 \wedge t < t_0 + 1 \rightarrow wait(t)) \wedge \\ & \quad \forall t > t_0 + 1\ \ \surd(t) \wedge \\ & \quad \forall t \leq t_0\ \forall a\in A\ (R(a, t) \leftrightarrow \exists e\in D\ a=r_1(e)) \wedge \\ & \quad R(wait, t_0) \wedge \\ & \quad \forall t\ \forall a\in A\ (t_0 < t < t_0+1 \rightarrow \neg R(a, t)) \wedge \\ & \quad \forall a\in A\ (R(a, t_0+1) \rightarrow a=s_2(d_0)) \wedge \neg R(wait, t_0+1) \\ & \vee \forall t > 0\ (wait(t) \wedge \forall a\in A\ (R(a, t) \leftrightarrow \exists e\in D\ a=r_1(e))).\end{aligned}$$

4. Let $a|b = c$. Then $cd(\int_{t\in(0,8)} a(t) \parallel \int_{v\in(7,9]} b(v)\ ) =$

$$\begin{aligned}& \forall t\ \forall d\in A\ (R(d, t) \rightarrow d=a \vee d=b \vee d=c) \wedge \\ & \exists t_1 \in (0,8]\ \exists t_2\in (7,9]\ \{((t_1 < t_2) \vee (t_1 = t_2 =8) \rightarrow \\ & \quad \forall t<t_1\ (wait(t) \wedge R(a,t) \wedge (t>7 \rightarrow R(b, t) \wedge R(c, t)) \wedge (t\leq 7 \rightarrow \forall d\in A\ (R(d, t) \rightarrow d=a)) \wedge \\ & \quad (t_1<8 \rightarrow a(t_1)) \wedge (t_1=8 \rightarrow \delta(t_1)) \wedge (7<t_1<8 \rightarrow R(b, t_1) \wedge R(c, t_1)) \wedge \\ & \quad (t_1\leq 7 \rightarrow \forall d\in A\ (R(d, t_1) \rightarrow d=a)) \wedge \\ & \quad (t_1<8 \rightarrow R(wait,t_1) \wedge \forall t \in (t_1, t_2)\ (wait(t) \wedge \\ & \qquad (t\leq 7 \rightarrow \forall d\in A\ \neg R(d, t) \wedge (t>7 \rightarrow R(b, t) \wedge \forall d\in A\ (R(d, t) \rightarrow d=b)) \wedge \\ & \qquad b(t_2) \wedge (t_2<9 \rightarrow R(wait,t_2) \wedge (t_2=9 \rightarrow \neg R(wait,t_2) \wedge \forall d\in A\ (R(d, t_2) \rightarrow d=b) \wedge \\ & \qquad \forall t>t_2\ \surd(t)) \\ & \qquad\quad \wedge (t_1 > t_2 \rightarrow \\ & \quad \forall t<t_2\ (wait(t) \wedge R(a, t) \wedge (t\leq 7 \rightarrow \forall d\in A\ (R(d, t) \rightarrow d=a)) \wedge (t>7 \rightarrow R(b, t) \wedge R(c, t)) \wedge \\ & \quad b(t_2) \wedge R(a, t_2) \wedge R(c, t_2) \wedge R(wait, t_2) \wedge \\ & \quad \forall t \in (t_2, t_1)\ (wait(t) \wedge R(a, t) \wedge \forall d\in A\ (R(d, t) \rightarrow d=a))) \wedge \\ & \quad (t_1<8 \rightarrow (a(t_1) \wedge R(wait, t_1) \wedge \forall d\in A\ (R(d, t_1) \rightarrow d=a)) \wedge \forall t>t_1\ \surd(t))) \wedge \\ & \quad (t_1=8 \rightarrow \delta(t_1))) \\ & \qquad\quad \wedge (t_1 = t_2 < 8 \rightarrow \\ & \quad \forall t<t_1\ (wait(t) \wedge R(a, t) \wedge (t\leq 7 \rightarrow \forall d\in A\ (R(d, t) \rightarrow d=a)) \wedge (t>7 \rightarrow R(b, t) \wedge R(c, t)) \wedge \\ & \quad c(t_1) \wedge R(a, t_1) \wedge R(b, t_1) \wedge R(wait,t_1) \wedge \\ & \quad \forall t>t_1\ \surd(t))\}.\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 $cd(P)$ is not necessary for proving a statement like $cd(P) \to \delta(8) \vee \exists t \in (0,8)\ a(t) \vee c(t)$. Even without such a proof system, our logic is valuable as a means to *express* such a statement.

6. RECURSION.

Now we consider recursive equations in this framework. Let us first consider the symbolic recursion equation $X = (a + b)\cdot X$. This equation has more than one solution in the symbolic ready trace model. First of all, there is the ready trace set given by:

$$\varphi \equiv \forall n \quad (a(n) \vee b(n)) \wedge \forall c \in A\ (R(c, n) \leftrightarrow c{=}a \vee c{=}b).$$

It is not hard to see that this formula determines a ready trace set V that is a solution of this equation. Another solution is the set of ready traces obtained by removing all ready traces with only finitely many a's or only finitely many b's:

$$V^* = V - \{\alpha : \{n : \alpha \textbf{ sat } a(n)\} \text{ is finite or } \{n : \alpha \textbf{ sat } b(n)\} \text{ is finite}\}.$$

Thus, this equation has at least two solutions in the ready trace model.

Next, we consider the real time recursion equation $X = \int_{t>0} a(t)\cdot X$. A first solution is the ready trace set determined by the formula:

$$\varphi_1 \equiv \forall t{>}0\ R(\mathrm{wait}, t) \wedge \forall b \in A\ (R(b, t) \leftrightarrow b{=}a) \wedge \forall t{>}0\ \{s{<}t : a(s)\} \text{ is finite.}$$

This formula determines a ready trace set that is a solution and consists only of non-Zeno and non-supertask traces. Next we consider the formula:

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

This also denotes a ready closed, time deterministic and right closed family of traces, and thus defines a process. 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. Next, we consider:

$$\varphi_3 \equiv \forall t{>}0\ ((R(\mathrm{wait}, t) \wedge R(a, t)) \vee \Omega(t)) \wedge \forall b \in A\ (R(b, t) \to b{=}a) \wedge$$
$$\forall t{>}0\ (\Omega(t) \to \exists t'{<}t\ \ \forall s{<}t'\ \exists r\ s{<}r{<}t' \wedge a(r)).$$

This formula determines another solution, a ready trace set that contains all ready traces given by φ_1, but moreover contains Zeno and supertask traces.

$$\varphi_4 \equiv \varphi_3 \wedge \forall t{>}0\ (\forall s{<}t\ \exists r\ s{<}r{<}t \wedge a(r) \to \Omega(t)).$$

This is like the previous case, but disallowing supertask traces.

We conclude that the equation $X = \int_{t>0} a(t)\cdot X$ comes nowhere near to having a unique solution.

6.1 TOPOLOGICAL PROCESS THEORY.

If we want that recursion equations nevertheless define a 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.

6.2 TRANSITION SYSTEMS.

Our proposal for the introduction of recursively defined processes in extensional ready trace theory is the following one, using techniques similar to [BEK88]. Let (taking the real time case) $\langle X(t_1, ..., t_k) \mid E\rangle$ 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 zero process, the fair merge operators and the φ □ operators in this section, as they are not considered in [BAB91].)

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 elaborate on this second step.

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

A *state* is a pair $\langle p, t\rangle$, where p is a closed process expression or the termination symbol √, 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 $t_1, t_2, ...$ 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 $t_i, t_{i+1}, ...$ 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 $t_n, t_{n+1}, ...$, 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 $\langle a(1), 0\rangle \xrightarrow{\text{wait}} \langle a(1), 1/2\rangle \xrightarrow{\text{wait}} \langle a(1), 3/4\rangle \xrightarrow{\text{wait}} \langle a(1), 7/8\rangle \xrightarrow{\text{wait}} ...$ 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., $s = \int_{0<t<1} a(t)$.)

6.2.2 DEFINITION. 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 $a \in A \cup \{wait\}$, if there exist s', t', s such that $\langle s', t'\rangle \xrightarrow{a} \langle s, t\rangle \in \sigma$ and $U = \{b \in A \cup \{wait\} \mid \exists s''\ \langle s', t'\rangle \xrightarrow{b} \langle s'', t\rangle \in TS\}$;
- $(U, wait)$ — if there exist $a \in A \cup \{wait\}, s', t', s'', t''$ such that $\langle s', t'\rangle \xrightarrow{a} \langle s'', t''\rangle \in \sigma$, $t' < t < t''$ and $U = \{wait\} \cup \{b \in A \mid \exists s\ \langle s', t'\rangle \xrightarrow{b} \langle s'', t\rangle \in TS\}$.

6.2.3 DEFINITION. 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 $\{RT(\sigma,TS) \mid \sigma$ is a path through TS and the source of the first transition of σ is $\langle p,0\rangle\}$.

6.2.4 CLAIM. Let p be a closed process expression without recursion. Then the ready trace set p defined above coincides with $[p]$, as defined in Section 4.2.2. (Note: it is especially interesting to check this claim for integrals over right-open intervals.)

6.3 COMMENTS.

1. In this way we obtain an infinite signature. For each expression $\langle X(t_1, ..., t_k) \mid E\rangle$ we have a process valued function in the signature.
2. There is *no* intention that unique solutions are obtained. Rather, we have 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.
3. A striking difference with untimed process algebra modulo bisimulation (in the absence of abstraction) seems to be that the very similar equations

e_1: $X = a \cdot X$ and

e_2: $X = \int_{t>0} a(t) \cdot X$ (or $X = \underline{a} \cdot X$)

behave so differently. Indeed, whereas e_1 has a unique solution in the bisimulation model, e_2 has no unique solution in the ready trace model. This motivates the notation $\langle X \mid e_2\rangle$ which indeed determines a ready closed, time deterministic and right closed trace set.

In fact, this matter derives from the fact that solving recursion equations in trace theory (with infinite traces) is in a set theoretic sense much more complicated than equation solving modulo bisimulation, because it involves the powerset construction or a restricted version of it to generate collections of schedulers. This also shows up with the untimed equation $X = (a + b) \cdot X$. To determine the set of infinite ready traces of this process involves some form of set theory. The construction of a transition system however is a combinatorial matter. The complexity of bisimulation theory in turn shows up if one realises that whether or not two transition systems both specified in untimed ACP using guarded specifications are bisimilar may easily be independent of ZF.

4. In [BEK86] we have outlined methods to avoid the introduction of infinite signatures and still deal with recursion equations that fail to have unique solutions. For example, instead of introducing $\langle X \mid X = \tau\cdot X + a\rangle$ one writes $\langle Y \mid Y = i\cdot Y + a\rangle$ (this Y is unique) and has $\tau_{\{i\}}(Y)$ satisfies $X = \tau\cdot X + a$. Using KFAR [BEK86] one then derives $\tau_{\{i\}}(Y) = \tau\cdot a$. In this way the notation $\langle X \mid X = \tau\cdot X + a\rangle$ can be avoided. We have not been able to find a similar way to avoid the $\langle X \mid E\rangle$ notation in the real time case.

5. The considerations above 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.

6. This approach is unable to deal with equations containing the fair merge and φ □ operators. The same holds to some extent for topological process theory, as the intended semantics can be an open set of ready traces.

7. 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 Section 6.7.

6.4 EXAMPLE. Let $E = \{X = a\cdot X, Y = b\cdot Y\}$. We notice that $P = \langle X \mid E\rangle \;{}_f\|_f\; \langle Y \mid E\rangle$ contains only traces executing infinitely many a's and infinitely many b's (exactly the ready trace set V^* defined in the beginning of this section). We can derive further that $\partial_{\{a\}}(P) = 0$.

6.5 EXAMPLE.

Let us consider the following process and property notations:

- $\mathcal{L}_{prop}(A) = \mathrm{RTL}\rho(A)$
- $\mathcal{L}_{proc}(A) = \mathrm{ACP}\rho\mathrm{I}(A)$ + conditionals + parametrisation of actions and processes by elements of a computable abstract data type specified by means of a complete term rewriting system.

Let $P = \langle P(\varnothing) \mid$

$$P(\sigma) = \int_{t>0} \sum_{d\in D} r_1(d)(t)\cdot P(\mathrm{enq}(d, \sigma)) \quad + \quad \delta(0) \lhd \mathrm{empty}(\sigma) \rhd \int_{t>0} s_2(\mathrm{deq}(\sigma))(t)\cdot P(\mathrm{tail}(\sigma)) \quad \rangle.$$

Here, σ denotes a queue over D, where D is a finite (data) set with fixed element d_0, and the datatype of queues is specified as follows:

$\varnothing$: Q
enq: $Q \times Q \to Q$
deq: $Q \to D$
tail: $Q \to Q$
empty: $Q \to \mathbb{B}$

$\mathrm{empty}(\emptyset) = \mathrm{T}$
$\mathrm{empty}(\mathrm{enq}(d, x)) = \mathrm{F}$
$\mathrm{deq}(\emptyset) = d_0$
$\mathrm{deq}(\mathrm{enq}(d, x)) = d \lhd \mathrm{empty}(x) \rhd \mathrm{deq}(x)$
$\mathrm{tail}(\emptyset) = \emptyset$
$\mathrm{tail}(\mathrm{enq}(d, x)) = x \lhd \mathrm{empty}(x) \rhd \mathrm{enq}(d, \mathrm{tail}(x))$.

$P(\emptyset)$ is one of the many forms of an unbounded queue. We expect (but failed to prove) that $P(\emptyset)$ has no complete description in RTL$\rho(A)$. ([KOY89b] proves such a fact for a temporal fragment of RTL$\rho(A)$ and a larger class of message passing systems which includes $P(\emptyset)$).

6.6 EXAMPLES.

We provide complete descriptions of some recursively defined processes. We look at the three clocks of [BAB91].

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

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

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

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

3. Whether or not the process $\langle C_3(1) \mid C_3(t) = \int_{v\in[t-0.01,t+0.01]} \mathrm{tick}(v)\cdot C_2(v+1)\rangle$ (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:

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

But higher order logic is not needed here. It is possible to encode the sequence $t_0, t_1, t_2, t_3, \ldots$ by one positive real number r. If we find an encoding such that the corresponding function decode, satisfying $\forall n \in \mathbb{N}\ \mathrm{decode}(r,n) = t_n$, is expressible in the arithmetic of the logic, then we can replace in the above description $\exists t_0, t_1, t_2, t_3, \ldots$ by $\exists r$, and each t_n by $\mathrm{decode}(r,n)$.

6.7 SUPERTASKS.

The normal task axiom (NTA) excludes supertasks: $NTA \equiv \forall t\ ((\forall s < t\ \exists r \in (s,t)\ \exists a \in A\ a(r)) \rightarrow \Omega(t))$, 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 8). We provide an operator that introduces supertask processes (i.e., ready trace sets that may contain supertasks).

The operator $\surd_\Omega$ is defined on ready traces by

$$\surd_\Omega(\alpha)(t) = \begin{cases} \alpha(t) & \text{if } \alpha(t) \neq \Omega \\ \surd & \text{if } \alpha(t) = \Omega. \end{cases}$$

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.

7. PROTOCOL SPECIFICATION: THE CONCURRENT ALTERNATING BIT PROTOCOL.

As an example of an application of ready trace theory we consider the Concurrent Alternating Bit Protocol (CABP) as described in [KOM90]. See Fig. 1. First of all, we consider symbolic ready trace theory.

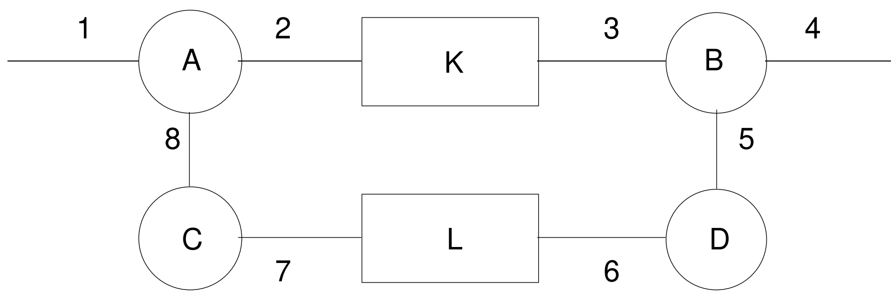

FIGURE 1.

Data from the finite data set D are to be transferred from port 1 to port 4, for acknowledgements through ports 2,3,6,7 an alternating bit from $B = \{0, 1\}$ is used, an acknowledgement at 5 or 8 is ac, and er is an error value. In the transmission channels, a choice is made between correct transmission (i), corrupted transmission (j) or loss of data (k).

7.1 SYMBOLIC SPECIFICATION.

We specify the components:

Data transmitter:

$$A = A^0$$

$$A^b = \sum_{d\in D} r_1(d)\cdot A^b_d \qquad b \in B$$

$$A^b_d = s_2(db)\cdot A^b_d + r_8(ac)\cdot A^{1\text{-}b} \qquad b \in B, d \in D.$$

Data transmission channel:

$$K = \sum_{f\in D\times B} r_2(f)\cdot(i\cdot s_3(f) + j\cdot s_3(er) + k)\cdot K \quad .$$

Data receiver:

$$B = B^0$$

$$B^b = (r_3(er) + \sum_{d\in D} r_3(d(1\text{-}b)))\cdot B^b + \sum_{d\in D} r_3(db)\cdot s_4(d)\cdot s_5(ac)\cdot B^{1\text{-}b} \qquad b \in B.$$

Acknowledgement transmitter:

$$D = D^1$$

$$D^b = r_5(ac)\cdot D^{1\text{-}b} + s_6(b)\cdot D^b \qquad b \in B.$$

Acknowledgement transmission channel:

$$L = \sum_{b\in B} r_6(b)\cdot(i\cdot s_7(b) + j\cdot s_7(er) + k)\cdot L.$$

Acknowledgement receiver:

$$C = C^0$$

$$C^b = (r_7(er) + r_7(1\text{-}b))\cdot C^b + r_7(b)\cdot s_8(ac)\cdot C^{1\text{-}b} \qquad b \in B.$$

Fairness assumption:

$$\varphi = \forall n\, \exists m > n\; R(i, m) \;\to\; \forall n\, \exists m > n\; i(m).$$

Encapsulation sets:

$$H(p,q) = \{s_p(x), r_p(x), s_q(x), r_q(x) : x \in D \cup D\times B \cup B \cup \{er, ac\}\}.$$

Specification of the protocol:

$$CABP = \partial_{H(5,8)}\big(\partial_{H(2,3)}(A \parallel \varphi \,\square\, K \parallel B)\ {}_f\!\parallel_f \partial_{H(6,7)}(C \parallel \varphi \,\square\, L \parallel D)\big).$$

Here, abusing notation, we write A instead of $\langle A \mid E\rangle$ (where E is the specification above) etc.

Now correctness of the protocol can be expressed as follows:

CABP **sat**

$$\forall n\, \forall d\in D\, (r_1(d)(n) \to \exists m{>}n\, (s_4(d)(m) \wedge \forall r \in (n, m)\, \forall e\in D\, (\neg R(r_1(e), r) \wedge \neg R(s_4(e), r)))) \wedge$$
$$\forall n\, \forall d\in D\, ((s_4(d)(n\text{-}1) \vee n{=}0) \to$$
$$\exists r{\ge}n\, (\forall v \in [n, r)\, \forall d\in D\, \neg R(r_1(d), v) \wedge \exists m{\ge}r\, (\forall v\in [r, m]\, \forall d\in D\, R(r_1(d), v) \wedge \exists e\in D\, r_1(e)(m) \wedge$$
$$\forall v \in [n, m]\, \forall d\in D\, \neg R(s_4(d), v)))).$$

7.2 Real time specification.

Data transmitter:

$$A = A^0$$

$$A^b = \int_{t>0} \sum_{d\in D} r_1(d)(t)\cdot A^b_d(t) \qquad b\in B$$

$$A^b_d(t) = s_2(db)(t+1)\cdot A^b_d(t+1) + \int_{v\le t+1} r_8(ac)(v)\cdot A^{1-b} \qquad b\in B, d\in D.$$

Data transmission channel:

$$K = \int_{t>0} \sum_{f\in D\times B} r_2(f)(t)\cdot(i(t+1)\cdot s_3(f)(t+2) + j(t+1)\cdot s_3(er)(t+2) + k(t+1)) \cdot K.$$

Data receiver:

$$B = B^0$$

$$B^b = \int_{t>0} (r_3(er)(t) + \sum_{d\in D} r_3(d(1-b))(t))\cdot B^b + \int_{t>0} \sum_{d\in D} r_3(db)(t)\cdot s_4(d)(t+1)\cdot s_5(ac)(t+2)\cdot B^{1-b} \qquad b\in B.$$

Acknowledgement transmitter:

$$D = D^1(0)$$

$$D^b(t) = \int_{v\le t+1} r_5(ac)(v)\cdot D^{1-b}(v) + s_6(b)(t+1)\cdot D^b(t+1) \qquad b\in B.$$

Acknowledgement transmission channel:

$$L = \int_{t>0} \sum_{b\in B} r_6(b)(t)\cdot(i(t+1)\cdot s_7(b)(t+2) + j(t+1)\cdot s_7(er)(t+2) + k(t+1))\cdot L.$$

Acknowledgement receiver:

$$C = C^0$$

$$C^b = \int_{t>0} (r_7(er)(t) + r_7(1-b)(t)) \cdot C^b + \int_{t>0} r_7(b)(t)\cdot s_8(ac)(t+1) \cdot C^{1-b} \qquad b\in B.$$

Fairness assumption:

$$\varphi = \forall t\, \exists s > t\, R(i, s) \rightarrow \forall t\, \exists s > t\, i(s).$$

Encapsulation sets are as in the symbolic case.

Specification of the protocol:

$$CABP = \partial_{H(5,8)}\big(\partial_{H(2,3)}(A \parallel \varphi \,\square\, K \parallel B) \parallel \partial_{H(6,7)}(C \parallel \varphi \,\square\, L \parallel D)\big).$$

Correctness of the protocol can be expressed as follows:

CABP **sat**

$$\forall t\, \forall d\in D\, (r_1(d)(t) \rightarrow \exists s>t\, (s_4(d)(s) \wedge \forall r \in (t, s)\, \forall e\in D\, (\neg R(r_1(e), r) \wedge \neg R(s_4(e), r)))) \wedge$$
$$\forall t\, \forall d\in D\, ((s_4(d)(t) \vee t=0) \rightarrow$$
$$\exists r>t\, (\forall v \in [t, r]\, \forall d\in D\, \neg R(r_1(d), v) \wedge ((\exists s>r\, (\forall v\in (r, s]\, \forall d\in D\, R(r_1(d), v) \wedge \exists e\in D\, r_1(e)(s) \wedge \forall v\in (t, s]\, \forall d\in D\, \neg R(s_4(d), v)))$$
$$\vee (\forall v>r\, \forall d\in D\, (R(r_1(d), v) \wedge wait(v)) \wedge \forall v>t\, \forall d\in D\, \neg R(s_4(d), v)))).$$

8. 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* (CAP): 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. 2.

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.

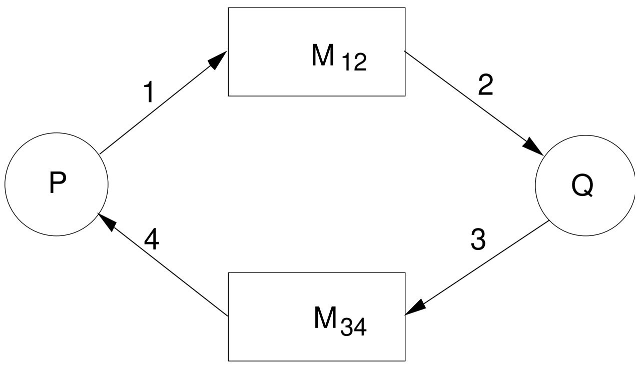

FIGURE 2.

8.1 COMMUNICATION.

We can describe the media as follows:

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

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

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

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

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. For simplicity, we shall not treat actions like a&b separately. For example, we can write X **sat** $a(t) \wedge b(t)$ instead of X **sat** a&b(t), violating the 'axiom' $\forall t\ \ a(t) \wedge b(t) \ \rightarrow\ a = b$.

8.2 SYNCHRONIZATION.
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 $\forall t{>}0\ \exists \alpha \in V_P\ \forall r{<}t\ \alpha(r)_2 = wait$ (where V_P denotes the ready trace set of P), but this statement cannot be rephrased as P **sat** φ, for any formula φ.

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 through the media are lost:

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

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

Now we can write the whole system as:

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

Due to the existence of the 0-process, this formulation works only under the additional assumption that for all t, $P(t) \neq 0$ (i.e., the ready trace set of P(t) is non-empty), and the same for Q.

8.3 THEOREM. For all processes P and Q satisfying the requirements above, R **sat** Φ_{syn} is false.
PROOF: By contradiction. Assume R **sat** Φ_{syn} for some P and Q satisfying the requirements above.

Let V_X denote the ready trace set of X, for X = P,Q,R,M. Let, for $\sigma_P \in V_P$, $t(\sigma_P)$ denote the unique time t such that σ_P **sat** p_attack(t), and similarly for Q. Then R **sat** Φ_{syn} implies

$$\forall r,s{>}0\ \forall \sigma_{Pi} \in V_{P\_init(r)}, \sigma_{Qi} \in V_{Q\_init(s)}, \sigma_P \in V_{r \gg P(r)}, \sigma_Q \in V_{s \gg Q(s)}, \sigma_M \in V_M \qquad (*)$$
$$(\sigma_{Pi}\cdot_{rt}\sigma_P) \parallel_{rt} \sigma_M \parallel_{rt} (\sigma_{Qi}\cdot_{rt}\sigma_Q) \in V_R \Rightarrow |t(\sigma_P) - t(\sigma_Q)| < 1.$$

That is, if σ_P and σ_Q are compatible, in the sense that their communication behavior is possible given the specification of M, then the times of attack specified by σ_P and σ_Q are at most one time unit apart. For each tuple $\sigma_P \in V_{r \gg P(r)}$, $\sigma_Q \in V_{s \gg Q(s)}$, $\sigma_M \in V_M$, we can define the number of useful communications as the number of send actions in σ_M before $max(t(\sigma_P),t(\sigma_Q))$. (Although $s_2(d)$

actions before time s and $s_4(d)$ actions before time r are not particularly useful, we count them.) Formally, $c(\sigma_P,\sigma_Q,\sigma_M) = k$ abbreviates

$\exists t_1 < \max(t(\sigma_P),t(\sigma_Q)) \ldots \exists t_k < \max(t(\sigma_P),t(\sigma_Q))$

$$\bigvee_{j=0..k} \bigwedge_{i=1..j} \left(\sigma_M \textbf{ sat } \exists d \in D\ s_2(d)(t_i) \wedge \bigwedge_{n=1..i-1} t_n \neq t_i\right) \wedge$$

$$\bigwedge_{i=j+1..n} \left(\sigma_M \textbf{ sat } \exists d \in D\ s_4(d)(t_i) \wedge \bigwedge_{n=j+1..i-1} t_n \neq t_i\right) \wedge$$

$$\forall t < \max(t(\sigma_P),t(\sigma_Q)) \left(\left(\bigwedge_{i=1..j} t \neq t_i\right) \rightarrow \sigma_M \textbf{ sat } \neg\exists d \in D\ s_2(d)(t)\right)$$

$$\forall t < \max(t(\sigma_P),t(\sigma_Q)) \left(\left(\bigwedge_{i=j+1..k} t \neq t_i\right) \rightarrow \sigma_M \textbf{ sat } \neg\exists d \in D\ s_4(d)(t)\right).$$

We now prove the following for each tuple $r,s>0$, $\sigma_{Pi} \in V_{P\_init(r)}$, $\sigma_{Qi} \in V_{Q\_init(s)}$, $\sigma_P \in V_{r\gg P(r)}$, $\sigma_Q \in V_{s\gg Q(s)}$, $\sigma_M \in V_M$ such that $(\sigma_{Pi}\cdot_{rt}\sigma_P) \parallel_{rt} \sigma_M \parallel_{rt} (\sigma_{Qi}\cdot_{rt}\sigma_Q) \in V_R$ and $|t(\sigma_P) - t(\sigma_Q)| < 1$:

1. $c(\sigma_P,\sigma_Q,\sigma_M) = k$ for some number k.
2. If $c(\sigma_P,\sigma_Q,\sigma_M) = k>0$, then there exists a tuple $r',s'>0$, $\sigma'_{Pi} \in V_{P\_init(r')}$, $\sigma'_{Qi} \in V_{Q\_init(s')}$, $\sigma'_P \in V_{r'\gg P(r')}$, $\sigma'_Q \in V_{s'\gg Q(s')}$, $\sigma'_M \in V_M$ such that $(\sigma'_{Pi}\cdot_{rt}\sigma'_P) \parallel_{rt} \sigma'_M \parallel_{rt} (\sigma'_{Qi}\cdot_{rt}\sigma'_Q) \in V_R$ and $c(\sigma'_P,\sigma'_Q,\sigma'_M) < k$.
3. If $c(\sigma_P,\sigma_Q,\sigma_M) = 0$, then there exists a tuple $r',s'>0$, $\sigma'_{Pi} \in V_{P\_init(r')}$, $\sigma'_{Qi} \in V_{Q\_init(s')}$, $\sigma'_P \in V_{r'\gg P(r')}$, $\sigma'_Q \in V_{s'\gg Q(s')}$, $\sigma'_M \in V_M$ such that $(\sigma'_{Pi}\cdot_{rt}\sigma'_P) \parallel_{rt} \sigma'_M \parallel_{rt} (\sigma'_{Qi}\cdot_{rt}\sigma'_Q) \in V_R$ and $|t(\sigma'_P) - t(\sigma'_Q)| \geq 1$.

Together, these three points contradict (*), because $V_R \neq \emptyset$.

1. Since the channels can only handle one message at a time, and each successful transmission takes 2 time units, there can be only finitely many send actions in σ_M in a finite period of time.

2. Let t_0 be the time at which the last $s_2(d)$ or $s_4(d)$ action occurs in σ_M before $\max(t(\sigma_P),t(\sigma_Q))$. Without loss of generality, we may assume this message is $s_2(d)$ for some $d \in D$. The new tuple is obtained as follows.

a. $r' = r$, $s' = s$, $\sigma'_{Pi} = \sigma_{Pi}$, and $\sigma'_{Qi} = \sigma_{Qi}$.

b. For $t \leq t(\sigma_P)$, $\sigma'_P(t) = \sigma_P(t)$. There is no essential change in σ_P. In particular $t(\sigma'_P) = t(\sigma_P)$.

c. For $t \leq t_0$, $\sigma'_M(t) = \sigma_M(t)$, except that $\sigma'_M(t_0 - 1)$ is i_2 instead of i_1 and $\sigma'_M(t_0)$ is $wait$ instead of $s_2(d)$, thus we turn the last useful communication into failure. As the following points do not introduce new communications before $t(\sigma_P)+1$, we have $c(\sigma'_P,\sigma'_Q,\sigma'_M) < k$ (since R **sat** Φ_{syn} implies $\max(t(\sigma'_P),t(\sigma'_Q)) < t(\sigma_P)+1$).

d. For $t < t_0$, $\sigma'_Q(t) = \sigma_Q(t)$. $\sigma_Q(t_0)$ was $r_2(d)$, but now the corresponding send action has disappeared. We cannot say much about the resulting behavior of Q. There are several possibilities:

- The read action by Q waits indefinitely.
- σ'_Q contains this $r_2(d)$ action, and it communicates with a later $s_2(d)$ action from M.
- σ'_Q contains this $r_2(d)$ action, at a time $t \geq t_0$, and it cannot communicate. In this case the encapsulation by H causes deadlock. This cannot mean that there is a deadlocked trace in V_R (it would contradict R **sat** Φ_{syn}, because the action q_attack does not occur). So it means that the deadlocked trace is 'overruled' by another one, i.e. σ'_Q has another action or $wait$ ready at time t.

(Here we need that $V_{s'\gg Q(s')}$ is right-closed, avoiding that an infinite set of traces, each overruling the previous one, would result in an empty trace set.) This gives us the last case:

- Q performs another action (possibly after waiting), and thereafter a completely different trace. This trace may involve $s_3(d)$ actions.

e. For $t_0 < t \leq t(\sigma_P)+1$, $\sigma'_M(t)$ is not $s_2(d)$ or $s_4(d)$. Attempts to communicate from P and Q are matched by corresponding read actions, but followed by i_2, respectively i_4, after one time unit. (In other words: $\sigma'_M(t) = \sigma_M(t)$, except that any 'new' attempts to communicate by Q are accepted, but not sent to P.) Note that this does not violate channel fairness, since the channel can handle only finitely many inputs in a finite amount of time. σ_M did not contain any $s_4(d)$ actions between t_0 and $t(\sigma_P)$, so this choice of σ'_M is consistent with b: for $t \leq t(\sigma_P)$, $\sigma'_P(t) = \sigma_P(t)$.

3. There is no communication between P and Q before $\max(t(\sigma_P),t(\sigma_Q))$. In this case we obtain the new tuple as follows.

a. $r' = r$ and $\sigma'_{P_i} = \sigma_{P_i}$.

b. $s' = t(\sigma_P)+1$, for $t < s'$: $\sigma'_{Q_i}(t) = (\{r_2(d) : d \in D, wait\}, wait)$ and
$\sigma'_{Q_i}(s') = (\{r_2(d) : d \in D, q\_arrive\}, q\_arrive) = \sigma_{Q_i}(s)$.

b. For $t \leq t(\sigma_P)+1$, $\sigma'_P(t) = \sigma_P(t)$ and $\sigma'_M(t) = \sigma_M(t)$. In particular $t(\sigma'_P) = t(\sigma_P)$.

c. $\sigma'_Q \in V_{s'\gg Q(s')}$, which may differ from $V_{s\gg Q(s)}$, but by assumption $V_{s'\gg Q(s')} \neq \emptyset$. Obviously, if $\sigma'_Q(t) = q\_attack$, then $t > s' = t(\sigma_P)+1$, thus $|t(\sigma'_P) - t(\sigma'_Q)| \geq 1$.

8.4 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:

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

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

Immediately after receiving an input, a new input can be received. Still, an unsuccessful communication takes 1 time unit, and a successful one 2 time units. We redefine M (and thus R) by:

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

We can get global synchronisation, if we allow supertasks (see Sections 5.5.3 and 6.7). 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, is 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.

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

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

8.5 THEOREM. If we do not allow supertasks, then for all processes P and Q satisfying the requirements of Section 8.2, R **sat** Φ_{syn} is false (recall that R is now defined using M^*_{12} and M^*_{34}).

PROOF: The proof is largely the same as that of Theorem 8.3. Since we do not allow supertasks, the number of useful communications between P and Q, as defined there, is again finite. More precisely, a Zeno trace would allow an infinite number of communications, but no attack actions following it.

This subtlety arises again in item 2e of the proof, where the trace σ'_M violates the fairness assumption of the channels, if σ'_Q perfoms an infinite number of send actions between t_0-2 and $t(\sigma_P)$-1: at least one of them should result in a send action in σ'_M between t_0 and $t(\sigma_P)$+1. Since $(t_0-2,t(\sigma_P)-1]$ is a finite interval, we must have a limit point t_1 in this interval with infinitely many sends in σ'_Q between t_1 and t_1 - 0.5. Consider another trace σ''_M of M that differs from σ'_M only in the fact that each send after t_1 - 0.5 is a successful one. This trace satisfies fairness, but because message transmission takes at least 1 time unit, Q cannot see the difference. Thus if we replace σ'_M by σ''_M, we have a trace of R in which Q performs an infinite number of send actions. Since we do not allow supertasks, this means that this trace does not satisfy Φ_{syn}, thus we conclude that σ'_Q cannot perfom an infinite number of send actions between t_0-2 and $t(\sigma_P)$-1.

8.6 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. This protocol was suggested by Jan Friso Groote [GRO92].

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

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

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

M and R are redefined accordingly.

Again, P and Q have asymmetric definitions. This time it is Q who periodically (every 8 time units) sends a message in order to determine if both processes are alive. When P receives such a message, it will time out after 7 time units. In order to make sure Q knows this, one succesful message exchange suffices. This exchange, initiated by P, will be done faster and faster, in order to fit within a 6 units time frame.

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

$$P(t) = \int_{v>t} r_4(^1/_2)(v) \cdot s_1(^1/_2)(v+^1/_2) \cdot P(v+^1/_2, 1/_2, 7+v) \quad \text{, where}$$

$$P(t, r, u) = r_4(r)(t+5r) \cdot p\_attack(u)) + s_1(^r/_2)(t + 6r) \cdot P(t+6r, ^r/_2, u)$$

(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.)

$$Q(t) = s_3(^1/_2)(t+8) \cdot Q(t+8) + \int_{v \le t+7} r_2(r)(v) \cdot s_3(r)(v+r) \cdot Q^*(v + 10r)$$

$$Q^*(u) = q\_attack(u+^1/_2) + \int_{v \le u} r_2(r)(v) \cdot s_3(r)(v+r) \cdot Q^*(u) \,.$$

We can clarify the protocol by looking at Fig. 3, in which P receives the first message from Q at (relative) time $-^1/_2$ and both processes time out at time $6^1/_2$. The figure shows all attempted communications, except the initial messages from Q. Q only attempts to reply if the corresponding message from P arrives. As soon as a pair of corresponding messages succeeds, P stops sending, and, as a consequence, so does Q.

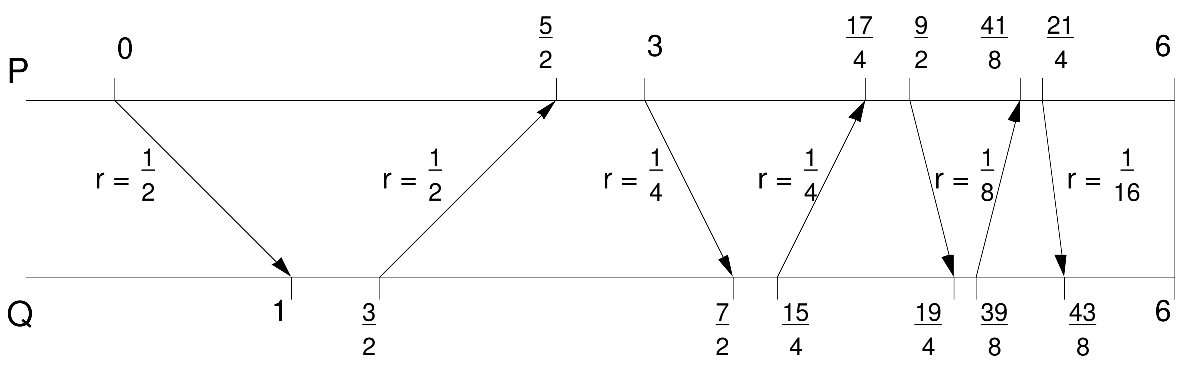

FIGURE 3.

9. CONCLUSIONS, PROBLEMS AND FUTURE WORK.

A number of problems remains 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 4 has grown into a large number of complex definitions, of which one would like to prove that they correspond to the intuitions behind them. Such a proof could compare a ready trace set defined as in Section 4 with the ready trace set of the transition sytem associated to the process term under consideration.

The decision to add a summand $\delta(2)$ to $\int_{1<t<2} b(t)$ seems counterintuitive. But giving an interpretation of terms containing an integral over an open interval requires a decision between the intuition that $\partial_{\{b\}}(a(1)\cdot\int_{1<t<2} b(t)) = a(1)\cdot\delta(2)$ (rather than 0) and the technical observation that the naive interpretation yields $a(1)\cdot\int_{1<t<2} b(t) + a(1)\cdot\int_{1<t<3} b(t) = a(1)\cdot\int_{1<t<3} b(t)$. 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 reasonable.

As it is defined here, Ready Trace Logic can only be used to describe properties of *all* traces of a process. In Section 8.2, we wanted to state that a process has a certain trace (for every $t>0$). Thus the usefulness of RTL 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 laws for the operators, for example: $\varphi+\varphi \neq \varphi$. Defining these connectives in terms of RTL 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 RTL, namely that its notation is very explicit. When used on a high level of specification, this is an advantage of RTL, 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 RTL is obviously crucial, before this higher level logic can be defined.

REFERENCES.

[BAB90] J.C.M. BAETEN & J.A. BERGSTRA, *Process algebra with a zero object,* in: Proc. CONCUR'90, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS 458, 1990, pp. 83-98.

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

[BAB92a] J.C.M. BAETEN & J.A. BERGSTRA, *Discrete time process algebra (extended abstract),* in Proc. CONCUR'92, Stony Brook (W.R. Cleaveland, ed.), Springer LNCS 630, 1992, pp. 401-420.

[BAB92b] J.C.M. BAETEN & J.A. BERGSTRA, *Real space process algebra,* report CSN 92/03, Dept. of Computing Science, Eindhoven University of Technology 1992 or report P9206, Programming Research Group, University of Amsterdam 1992. To appear in Formal Aspects of Computing.

[BABK86] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP, *Syntax and defining equations for an interrupt mechanism in process algebra,* Fund. Inf. IX (2), 1986, pp. 127-168.

[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.

[BAW90] J.C.M. BAETEN & W.P. WEIJLAND, *Process algebra,* Cambridge Tracts in Theor. Comp. Sci. 18, Cambridge University Press 1990.

[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.

[BEK86] J.A. BERGSTRA & J.W. KLOP, *Verification of an alternating bit protocol by means of process algebra,* in: Math. Methods of Spec. and Synthesis of Software Systems '85 (W. Bibel & K.P. Jantke, eds.), Springer LNCS 215, 1986, pp. 9-23.

[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.

[KOM90] C.P.J. KOYMANS & J.C. MULDER, *A modular approach to protocol verification using process algebra,* in: Applications of Process Algebra (J.C.M. Baeten, ed.), Cambridge Tracts in Theor. Comp. Sci. 17, Cambridge University Press 1990, pp. 261-306.

[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.

[MAP92] Z. MANNA, A. PNUELI*, The temporal logic of reactive and concurrent systems,* Springer Verlag 1992.

[MEY85] J.-J. CH. MEYER, *Merging regular processes by means of fixed point theory,* TCS 45, 1985, pp. 193-260.
[MIL89] R. MILNER, *Communication and concurrency,* Prentice Hall International 1989.
[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.