Constraint Orientated Specification with CSP and Real Time Temporal Logic

Justin Pearson
Department of Computer Systems
Box 325
Uppsala University
Sweden
justin@docs.uu.se

Jeremy Bryans
Computing Laboratory
University of Kent at Canterbury CT2 7NF
J.W.Bryans@ukc.ac.uk

July 22, 1998

1 Introduction

A popular specification style, particularly for the initial specification of a system, is the *constraint-oriented style*, where the constraints are properties required to hold of the final system. This style is independent of the particular specification notation being used: properties are individually described, and then composed (using, for example, parallel composition within process-algebraic notations, or conjunction within logic-based languages) to form an initial description of the system. This initial description can then be refined down to a specification suitable for implementation, using the refinement principles and rules offered by the chosen notation. The appeal of this style lies in the fact that the initial specification merely asserts properties of the system, and makes no demands on how the system should be implemented. So the specifier may gain confidence in the specification by, for example, checking the logical consistency, and deducing further consequences of the specification, before any consideration is given to implementation. Implementation decisions are made during the refinement steps, in which choices left open by the specification are refined away.

One of the difficulties of using this style is that, for any particular notation, certain types of constraints are much harder to capture than others. For example, a language which captures abstract behavioural constraints may not be able to capture so easily specific timing constraints. A logical extension of the constraint-oriented style would therefore be to allow the specifier to capture individual constraints in any of a range of languages, and to provide a formal semantic framework for combining these constraints and performing the necessary consistency checking.

In this paper we propose such a specification framework, which allows the specifier a choice of two languages: Communicating Sequential Processes (CSP) [Hoa85] and a version of Propositional Temporal Logic (PTL), derived from [Eme90].

CSP is a process-algebraic language designed for the specification and analysis of parallel systems and (our version of) PTL is a real-time temporal logic designed to capture time-dependent constraints concisely. The behaviour of a CSP process is dependent on its environment; it is therefore difficult to assert global properties. PTL can be easily used to express global timing properties of systems, but it is less suited to describing the purely behavioural aspects. We will therefore develop a framework in which a specification is a pair (P, ϕ), where P is a CSP process and ϕ is a formula of PTL. Global and timing constraints can be described within PTL, and behavioural constraints can be described within CSP.

Both components of a specification have to be checked for mutual consistency, to do this we present a common semantic framework for both PTL and CSP. Since we wish to retain all the behavioural and all the timing information in this mapping, we choose to use a real-time CSP semantic model. However, the existing real-time CSP models [Ree88, Sch92, Dav93] insist that recursive processes must be *time-guarded*, that is some time must elapse between any instantiation of a process and its recursive invocation. This facilitates the task of semantically defining recursive processes, but goes against the philosophy of the dual language style which we develop here. We therefore present in Section 2 a novel denotational model for CSP (which we call $\mathcal{M}_U$, for unguarded CSP), which does not require recursive processes to be time-guarded.

In Section 3 we present the logic PTL, and define a satisfaction relation between PTL statements CSP processes, to allow us to determine whether processes and formulae are consistent.

In Section 4 we present the definition of a specification pair, give a refinement relation between specification pairs, and show how this refinement relation is related to deduction within PTL and the common CSP notion of refinement. This is illustrated with a simple example.

The main contribution of this paper is therefore the framework of specification pairs, and further contributions are the development of a new CSP semantic model and the interpretation of PTL formulae over this model.

2 The CSP model

Communicating Sequential Processes (CSP) [Hoa85] is a process-algebraic language designed for the specification and analysis of parallel systems. It is a language of *processes* and *events*: processes describe patterns of events, and may be built up and combined using a set of powerful operators. The language of CSP has been extended to include timing constructs, and a number of semantic models for real-time CSP have been proposed [RR86, RR87, Sch92]. All these models require that processes be *time-guarded* — some time must elapse between any instantiation of a process and its recursive invocation. Therefore all recursive processes must contain some timing information. Consider, for example, a simple behavioural constraint as captured by the CSP process

$$Light = on \to off \to Light$$

which requires that the light may initially switched on, and then may be switched on and off alternately. If we wish to combine this constraint with one which explicitly mentions timing values, we must first interpret it within a timed semantic model. But then we must insert a delay somewhere in the process, which was not part of the original constraint. This is contrary to our desire to separate as much as possible the concerns of timing and behaviour.

We therefore present in this section a new denotational semantic model $\mathcal{M}_U$, which allows recursive processes to contain no timing information. In doing so we build on the work on unbounded non-determinism presented in [Ros93] and [Sch92], and the work on fixed points of recursive processes presented in [MRS95].

2.1 Notation

We assume a set of actions Σ. A *timed action* is a pair (t, a), where $t \in \mathbb{R}^+$ and $a \in \Sigma$. A *timed trace* is a sequence of chronologically ordered timed actions. Timed traces may be finite or infinite. The set of timed traces is denoted TT.

A *timed refusal* is a set of timed actions. Each timed refusal is associated with a timed traces, and record the events which the process was seen to refuse while performing that trace. The set of all refusal sets is denoted $RSET$.

A *divergence value* is a non-negative real number, or the symbol ∞. This records the time at which the process was seen to have diverged.

An *observation* is a triple (s, X, d), where s is timed trace, X is a timed refusal and d is a divergence value.

Some useful operators on traces and refusals are:

$s \uparrow I$ extracts from the timed trace s only the timed actions with times in the interval I, where I may be closed or right-open. It is defined as

$$\begin{aligned} \langle\rangle \uparrow I &= \langle\rangle \\ (t, a) \frown s \uparrow I &= \begin{cases} (t, a) \frown (s \uparrow I) & \text{if } t \in I \\ s \uparrow I & \text{otherwise} \end{cases} \end{aligned}$$

We overload the extraction operator by defining it on refusals: $X \uparrow I$ extracts from the timed refusal X only the timed actions with times in the interval I, where I may be closed or right-open.

$$X \uparrow I = X \cap \Sigma \times I$$

And as convenient abbreviations:

$$\begin{aligned} &s, X \upharpoonright t = s, X \uparrow [t, \infty) \\ &s, X \upharpoonleft t = s, X \uparrow [0, t) \end{aligned}$$

The beginning of a trace is defined as:

$$\begin{aligned} &\text{begin}(t, a) \frown s = t \\ &\text{begin}\langle\rangle = \infty \end{aligned}$$

and the end of a trace as

$$\begin{aligned} &\text{end}\, s \frown (t, a) = t \\ &\text{end}\langle\rangle = 0 \end{aligned}$$

The beginning and end of a refusal set are defined as

$$\begin{aligned} &\text{begin}\, X = \min\{t \mid \exists\, a \bullet (t, a) \in X\}, \text{ if } X \neq \emptyset \\ &\text{begin}\, \emptyset = \infty \\ &\text{end}\, X = \max\{t \mid \exists\, a \bullet (t, a) \in X\}, \text{ if } X \neq \emptyset \\ &\text{end}\, \emptyset = 0 \end{aligned}$$

The begin and end operators can be extended to whole observations:

$$\begin{aligned}
\text{begin}(s, X, d) &= \min(\text{begin}\, s, \text{begin}\, X, d) \\
\text{end}(s, X, d) &= \max(\text{end}\, s, \text{end}\, X, d), \text{ if } d \text{ is finite}, \\
&\quad \max(\text{end}\, s, \text{end}\, X), \text{otherwise.}
\end{aligned}$$

We overload the subtraction operator (the addition operator is overloaded in an analogous way), to allow us to subtract time values from traces and refusals.

$$\begin{aligned}
\langle\rangle - t &= \langle\rangle \\
((u, a) \frown s) - t &= (u - t, a) \frown (s - t), \quad \text{if } u \geqslant t \\
((u, a) \frown s) - t &= s - t, \qquad\qquad\qquad \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\emptyset - t &= \langle\rangle \\
X - t &= \{(u, a) \mid (u + t, a) \in X \wedge u \geqslant 0\}
\end{aligned}$$

$$((u, a) \frown s) + t = (u + t, a) \frown s$$

$$X + t = \{(u + t, a) \mid (u, a) \in X\}$$

Finally, we say that a trace s is a prefix of a trace t if the trace s can be extended to the trace t. Formally, we write this as

$$s \leqslant t \Leftrightarrow \exists\, u \bullet s \frown u = t$$

2.2 Syntax

The syntax used in this paper is similar to real-time CSP. The significant differences are: we include a second prefix operator ($\rightarrowtail$), and to simplify the presentation, do not include message passing or variables.

If P is a process, t is a time value, I is an index set, A is a set of events and a is a single event, then the syntax used in this paper is

$$\begin{aligned}
P ::= \bot \mid \mathit{Stop} \mid \mathit{Skip} \mid \mathit{Wait}(t) \mid P \,|[\, A \,]|\, P \mid P \,\Box\, P \mid \\
\textstyle\bigsqcap_{i \in I} P \mid a \rightarrow P \mid a \mapsto P \mid P \,;\, P \mid P \rhd\{t\}\, P \mid P \setminus A
\end{aligned}$$

These represent: the most nondeterministic process, $\bot$; the broken process, Stop; successful termination, Skip; delay, $\mathit{Wait}(t)$; parallel composition, $P \,|[\, A \,]|\, P$; external choice, $P \,\Box\, P$; nondeterministic choice, $\bigsqcap_{i \in I} P$; first form of action prefix, $a \rightarrow P$; second form of action prefix, $a \mapsto P$; sequential composition, $P \,;\, P$; timeout, $P \rhd\{t\}\, P$ and hiding, $P \setminus A$.

2.3 The axioms

A process S is a subset of $\mathcal{O}_U$, where $\mathcal{O}_U = TT \times TREF \times \mathbb{R}^+ \cup \{\infty\}$, which satisfies the following axioms.

Axiom 1 The empty observation is an observation of any process.

$$(\langle\rangle, \emptyset, \infty) \in S$$

Axiom 2 Observations are prefix closed.

$$\begin{array}{l} (s \frown s', X, d) \in S \Rightarrow \forall\, t \in [\mathrm{end}\, s, \mathrm{begin}\, s'] \bullet \\ \qquad\qquad r < d \Rightarrow (s, X \uparrow [0, t), \infty) \in S \\ \qquad\qquad t = d \Rightarrow ((s, X \uparrow [0, t), \infty) \in S \\ \qquad\qquad\qquad\quad \wedge \\ \qquad\qquad\qquad\quad (s, X \uparrow [0, t), d) \in S)) \\ \qquad\qquad t > d \Rightarrow (s, X \uparrow [0, t), t) \in S \end{array}$$

Axiom 3 Refusal information is subset closed.

$$(s, X \cup Y, d) \in S \Rightarrow (s, X, d) \in S$$

Axiom 4 Maximal refusal sets exist.

$$\begin{array}{l} (s, X, \infty) \in S \Rightarrow \exists\, X' \bullet \\ \qquad\qquad (s, X', \infty) \in S \wedge X \subseteq X' \wedge \\ \qquad\qquad \forall (t, a) \in \mathbb{R}^+ \times \Sigma \bullet (t, a) \notin X' \Rightarrow \\ \qquad\qquad\qquad (s \uparrow [0, t] \frown (t, a), X' \uparrow [0, t), \infty) \in S \end{array}$$

From this axiom, taken together with the chaos axiom below, we can derive two important properties of observations within a CSP process. The first is that all observations may be extended [1].

Definition 1 An observation (s', X', d') is an *extension* of an observation (s, X, d), written $(s', X', d') \geqslant_E (s, X, d)$ if

$s \leqslant s'$
$X \subseteq X'$
$d \geqslant d'$ if d is finite
$d' \geqslant \mathrm{end}(s, X, d)$ if d is infinite

[1] This is similar to the behavioural partial order in [Sch92].

This is essentially an information ordering on observations. Information may be added to an observation in two ways. The first is by *filling out* the information already contained, either by adding refusal information or by improving divergence information, and the second is by increasing the duration of the observation, by adding further trace, refusal or divergence information.

A consequence of the above definition is the existence of point-wise maximal observations. A point-wise maximal observation is one which contains complete information for every point in time, up to the end of the observation. Therefore the only way to add information to these observations is by increasing their duration.

Definition 2 (s, X, d) is *point-wise maximal* in $[\![P]\!]$ whenever

$$\forall (s', X', d') \in [\![P]\!].(s', X', d') \geqslant_E (s, X, d) \Rightarrow \begin{array}{l}(s', X', d') = (s, X, d) \\ \quad\vee \\ \mathrm{end}(s', X', d') > \mathrm{end}(s, X, d)\end{array}$$

Axiom 5 Divergence is chaotic. If an observation contains a divergence value of d, where $d \neq \infty$, this means that the process diverged at or before that time. This is consistent with the CSP philosophy of including all possible observations after divergence.

$$\forall\, d \neq \infty \bullet (s, X, d) \in S \Rightarrow (s \frown (s' + d), X \cup (X' + d), d' + d) \in S$$

The final axiom is a requirement that every nondivergent process may be implemented deterministically, and furthermore that the process is equal to the union of these implementations. Before we present it, we require some definitions.

Definition 3 Refinement is defined on processes as $Q \sqsubseteq P$ (Q is refined by P) if and only if

$$\forall (s, X, d) \in P \bullet (s, X, d) \in Q$$

or

$$[\![P]\!] \subseteq [\![Q]\!]$$

The lowest member of this order is $\perp$, the highest are those that cannot be further refined.

Definition 4 An observation $(s, X, d) \in P$ is an *earliest diverging observation* in P, if $d = \min\{d' \mid (s, X, d') \in [\![P]\!]\}$.

Predeterministic processes are deterministic until they diverge, if they do.

Definition 5 Process P is predeterministic if for all earliest diverging observations

$$\begin{array}{l} (s, X, d) \in P, (t, a) \in X \bullet \\ \qquad t < d \Rightarrow (s \uparrow [0, t] \frown \langle (t, a) \rangle, X \uparrow [0, t), \infty) \notin P \end{array}$$

Axiom 6 The final axiom is then defined as

$$imp(P) \neq \emptyset \wedge P = \bigcup imp(P)$$

where $imp(P)$ is the set of predeterministic implementations of P, defined as

$$imp(P) = \{Q \mid P \sqsubseteq Q \wedge Q \text{ is predeterministic}\}$$

Alternatively the last axiom can be stated in terms of upward closure:

Definition 6 The upward closure of a set of observations in S is defined as

$$\overline{S} = \{(s, X, d) \mid \forall t \bullet (s, X, d) \upharpoonright t \in S\}$$

That is, if all finite prefixes of an infinite observation are in S, then the infinite observation is in $\overline{S}$.

The last axiom simply states that predeterministic processes are equal to their own closure.

2.4 The Equations

The semantics of a CSP process is given by $\mathcal{F}_U$, where $\mathcal{F}_U$ is a function from CSP syntax to $\mathcal{M}_U$. It is defined inductively over the CSP operators as

$$\mathcal{F}_U[\![\perp]\!] = \{(s, X, d) \mid s \in TT \wedge X \in RSET \wedge d \in \mathbb{R}^+ \cup \{\infty\}\}$$

$$\mathcal{F}_U[\![Stop]\!] = \{(\langle\rangle, X, \infty) \mid X \in RSET\}$$

$$\begin{array}{rl} \mathcal{F}_U[\![\, Wait(d)]\!] = & \{(\langle\rangle, X, \infty) \mid \checkmark \notin X \uparrow [d, \infty)\} \\ & \cup \\ & \{(\langle (t_\checkmark, \checkmark) \rangle, X, \infty) \mid t_\checkmark \geqslant d \wedge \checkmark \notin X \uparrow [d, t_\checkmark)\} \end{array}$$

$$
\begin{aligned}
\mathcal{F}_U[\![P \,|[\,A\,]|\, Q]\!] = \{(s, X, d) \mid\ & \exists\, s_P, X_P, d_P, s_Q, X_Q, d_Q \bullet \\
& (s_P, X_P, d_P) \in \mathcal{F}_U[\![P]\!] \wedge \\
& (s_Q, X_Q, d_Q) \in \mathcal{F}_U[\![Q]\!] \wedge \\
& s \in s_P \,|[\,A\,]|\, s_Q \wedge \\
& X = X_P \,|[\,A\,]|\, X_Q \wedge \\
& d = \min\{d_P, d_Q\}\}
\end{aligned}
$$

where $s_P \,|[\,A\,]|\, s_Q$ is defined recursively as (assuming $a \in A;\ b, c \notin A$)

$$
\begin{aligned}
\langle\rangle \,|[\,A\,]|\, \langle\rangle &= \{\langle\rangle\} \\
s_P \,|[\,A\,]|\, \langle\rangle &= \{s_P\} \\
\langle\rangle \,|[\,A\,]|\, s_Q &= \{s_Q\} \\
(t, a) ^\frown s_P \,|[\,A\,]|\, (t, a) ^\frown s_Q &= \{(t, a) ^\frown (s_P \,|[\,A\,]|\, s_Q) \\
(t_1, b) ^\frown s_P \,|[\,A\,]|\, (t_2, c) ^\frown s_Q &= \{(t_1, b) ^\frown (s_P \,|[\,A\,]|\, (t_2, c) ^\frown s_Q)\}, \text{ if } t_1 < t_2 \\
&= \{(t_2, c) ^\frown ((t_1, b) ^\frown s_P \,|[\,A\,]|\, s_Q)\}, \text{ if } t_2 < t_1 \\
&= \{(t_1, b) ^\frown (s_P \,|[\,A\,]|\, (t_2, c) ^\frown s_Q)\} \\
&\quad\ \cup \\
&\quad\ \{(t_2, c) ^\frown ((t_1, b) ^\frown s_P \,|[\,A\,]|\, s_Q)\}, \text{ if } t_1 = t_2
\end{aligned}
$$

The recursive definition is necessary because the order of simultaneous events is important.

$X_P \,|[\,A\,]|\, X_Q$ is defined as the unique X such that

$$
\begin{aligned}
& X \restriction A = X_P \restriction A \cup X_Q \restriction A \\
& \wedge \\
& X \restriction (\Sigma \setminus A) = X_P \restriction (\Sigma \setminus A) \cap X_Q \restriction (\Sigma \setminus A)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}_U[\![P \,\Box\, Q]\!] = &\ \{(s, X, d) \mid (s, X, d) \in \mathcal{F}_U[\![P]\!] \wedge \\
&\qquad (\langle\rangle, X \uparrow [0, \text{begin}\, s), \infty) \in \mathcal{F}_U[\![Q]\!]\} \\
&\ \cup \\
&\ \{(s, X, d) \mid (s, X, d) \in \mathcal{F}_U[\![Q]\!] \wedge \\
&\qquad (\langle\rangle, X \uparrow [0, \text{begin}\, s), \infty) \in \mathcal{F}_U[\![P]\!]\}
\end{aligned}
$$

$$
\mathcal{F}_U[\![P \sqcap Q]\!] = \mathcal{F}_U[\![P]\!] \cup \mathcal{F}_U[\![Q]\!]
$$

$$
\begin{aligned}
\mathcal{F}_U[\![a \mapsto P]\!] = {} & \{(\langle\rangle, X, \infty) \mid a \notin \alpha(X)\} \\
& \cup \\
& \{(s, X, d) \mid \exists\, s_P, X_P, d_P, t_a \bullet \\
& \qquad (s_P, X_P, d_P) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \qquad s = \langle (t_a, a)\rangle \frown s_P \wedge \\
& \qquad a \notin \alpha(X \uparrow [0, t_a)) \wedge \\
& \qquad X - t_a = X_P \wedge \\
& \qquad d = d_P + t_a\}
\end{aligned}
$$

The new operator $\rightarrow$ includes nondeterministic waits on either side of the action a.

If t_o is the time at which a is offered, t_a the time at which it is accepted, and t_P the time at which the process P is initiated, then the definition is:

$$
\begin{aligned}
\mathcal{F}_U[\![a \rightarrow P]\!] = {} & \{(\langle\rangle, X, \infty) \mid \exists\, t_o \in \mathbb{R}^+ \bullet [t_o, \infty) \times \{a\} \cap X = \emptyset\} \\
& \cup \\
& \{((t_a, a) \frown s, X, d) \mid (s_P, X_P, d_P) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \qquad \exists\, t_o \leqslant t_a \bullet [t_o, t_a) \times \{a\} \cap X = \emptyset \wedge \\
& \qquad \exists\, t_P \geqslant t_a \bullet s = s_P + t_P \wedge \\
& \qquad\qquad X - t_P = X_P \wedge \\
& \qquad\qquad d = d_P + t_P\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}_U[\![\mathit{Skip}]\!] = {} & \{(\langle\rangle, X, \infty) \mid \checkmark \notin \alpha(X)\} \\
& \cup \\
& \{(\langle (t_\checkmark, \checkmark)\rangle, X, \infty \mid \checkmark \notin \alpha(X \uparrow [0, t_\checkmark))\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}_U[\![P\ ;\ Q]\!] = {} & \{(s, X, d) \mid (s, X \cup [0, \min\{\mathrm{end}(s, X), d\}) \times \{\checkmark\}, d) \in \mathcal{F}_U[\![P]\!]\} \\
& \cup \\
& \{(s, X, d) \mid \exists\, s_P, X_P, s_Q, X_Q, d_Q \bullet \\
& \qquad (s_P \frown \langle (t_\checkmark, \checkmark)\rangle, X_P \cup [0, t_\checkmark) \times \{\checkmark\}, \infty) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \qquad (s_Q, X_Q, d_Q) \in \mathcal{F}_U[\![Q]\!] \wedge \\
& \qquad s = s_P \frown (s_Q + t_\checkmark) \wedge \\
& \qquad X = X_P \cup (X_Q + t_\checkmark) \wedge \\
& \qquad d = d_Q + t_\checkmark\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}_U[\![P \triangleright\{t\}\ Q]\!] = {} & \{(s, X, d) \mid (s, X, d) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \qquad \min\{d, \text{begin}\, s\} \leqslant t\} \\
& \cup \\
& \{(s, X, d) \mid \exists\, s_Q, X_Q, d_Q \bullet \\
& \qquad (s_Q, X_Q, d_Q) \in \mathcal{F}_U[\![Q]\!] \wedge \\
& \qquad (\langle\rangle, X \uparrow [0, t), \infty) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \qquad s = s_Q + t \wedge \\
& \qquad X - t = X_Q \wedge \\
& \qquad d = d_Q + t\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}_U[\![P \setminus A]\!] = \{(s, X, d) \mid {} & \exists\, s_P, X_P \bullet \\
& (s_P, X_P, d) \in \mathcal{F}_U[\![P]\!] \wedge \\
& s = s_P \setminus A \wedge \\
& X_P = X \cup [0, \min\{\text{end}(s, X), d\}) \times A\}
\end{aligned}
$$

2.5 Recursion

None of the above clauses explicitly introduce finite divergence values into the observations of a process. We can only create divergent processes using recursion. The semantics of a process $X = F(X)$, where F is a CSP process, is given by the least fixed point of the corresponding function over the domain $\mathcal{M}_U$. As an example, consider the process $P = a \to P$. The traces generated by this process are

$$\langle\rangle, \langle(t_1, a)\rangle, \ldots \langle(t_i, a)\rangle^n, \ldots$$

If the trace becomes infinitely long at a finite time d, a divergence value of d is introduced.

The exact semantics of the process $P = a \to P$ is therefore given by

$$
\begin{aligned}
& \{(s, X, \infty) \mid \alpha(s) = \{a\} \wedge \\
& \qquad a \notin \alpha(X) \wedge \\
& \qquad \#(s) = \infty \Rightarrow \text{end}(s) = \infty\} \\
& \cup \\
& \{(s, X, d) \mid \exists\, t \leqslant d \bullet \#(s \uparrow [0, t] \restriction a) = \infty \wedge \\
& \qquad \forall\, t' < t \bullet \#(s \uparrow [0, t') < \infty) \wedge \\
& \qquad\qquad \alpha(s \uparrow [0, t') = \{a\}) \wedge \\
& \qquad\qquad a \notin (\alpha(X) \uparrow [0, t'))\}
\end{aligned}
$$

Standard CSP recursion theory [Hoa85, Ros88] cannot be applied, because the model $\mathcal{M}_U$ is not a complete partial order (CPO). Further the metric space theory of ordinary real-time CSP [Dav93] cannot be used, because the metric is not applicable. To see that $\mathcal{M}_U$ is not a CPO, consider the set of processes defined by

$$P_n = \sqcap_{n \leqslant t < \infty} Wait(t) \ ; \ a \to Stop$$

The chain $\{P_i\}_{i=0}^{\infty}$does not have an upper bound. Any upper bound would have to refuse a at all times, since for any time t we can always find a process which will refuse a at that time, but would not be able to refuse it forever, since no individual component can. This is forbidden by the maximal refusal set axiom: an action must be either possible or refusable at all times.

We therefore use the framework presented in [MRS95], which relies on *local cpo's*.

Definition 7 A *local cpo* is a partially ordered set with a least element, in which every directed subset with an upper bound has a least upper bound.

A local cpo is much weaker than a cpo. Obviously every cpo is a local cpo, but in a local cpo we only require least upper bounds to exist when upper bounds exist (in a cpo least upper bounds would have to exist for every increasing chain).

Theorem 1 $\mathcal{M}_U$ with the nondeterminism ordering is a local cpo: it has a least element, $\perp$, and every directed subset with an upper bound has a least upper bound.

Proof 1 Let S be a subset of $\mathcal{M}_U$, directed under the nondeterminism ordering $\sqsubseteq$, and suppose that an upper bound of S exists. Consider $\bigcup\{S' \mid S' \textit{ is an upper bound for } S\}$. This is non empty, since S has at least one upper bound, and it must be the least upper bound, since it contains all other upper bounds. It is also a legitimate process, since it satisfies all the process axioms.

Not all monotone functions over local cpos have fixed points. To see why this is so, consider the local cpo defined on the set $Q = \{\frac{n-1}{n} \mid n > 0\}$ by the mapping $f(\frac{n-1}{n}) = \frac{n}{n+1}$. The monotone function f maps each element to the one above it, but clearly has no fixed point.

In CSP terms, this means that not all monotonic chains of processes will have fixed points. For example, consider the chain of processes defined by:

$$P_n = \sqcap_{t \in [1-\frac{1}{n},1)} Wait(t) \ ; \ a \to Stop$$

The limit of this chain must refine each element in the chain. Thus, in the limit process, a cannot be offered at any time before 1, because for any time before 1 we can find a P_i which does not offer a at that time. But on the other hand a must be offered before time 1, since all elements of the chain offer a before time 1. This contradiction ensures that the limiting process does not exist.

A sufficient condition for a monotone function over a local cpo to have a fixed point is for it to have a *pre-fixed point*.

Definition 8 A pre-fixed point of a function f over a set X is an element $x \in X$ such that $f(x) \leqslant x$.

If we can show that a pre-fixed point exists, then the following theorem shows that we can find a fixed point.

Theorem 2 X is a local cpo with least element $\perp$, and $f : X \to X$ is a monotone function.

If x is a pre-fixed point of f, then f has a least fixed point given by

$$\mathit{fix}(f) = \sup\{f^{\alpha}(\perp) \mid \alpha \,\text{an ordinal}\}$$

Proof 2 Omitted, see [MRS95].

Theorem 2 only proves the existence of a fixed point for functions which already have a pre-fixed point, i.e. a point x such that $f(x) \leqslant x$.

In order to prove the existence of this pre-fixed point, we use a dominated convergence theorem.

Theorem 3 Let Q be a local cpo, E a set, $i : E \to Q$ a function and $f : Q \to Q$ a monotone selfmap. Suppose there is a related function $\overline{f}$ such that $f \circ i \leqslant i \circ \overline{f}$, i.e. the following diagram commutes up to $\leqslant$.

$$\begin{tikzcd} E \arrow[r, "\overline{f}"] \arrow[d, "i"'] \arrow[dr, phantom, "\leqslant"] & E \arrow[d, "i"] \\ Q \arrow[r, "f"'] & Q \end{tikzcd}$$

If $x \in E$ is a fixed point of $\overline{f}$, then $i(x) \in Q$ is a pre-fixed point of f.

Proof 3 $f(i(x)) \leqslant i(\overline{f}(x))$, and since $x = \overline{f}(x)$ we conclude that $f(i(x)) \leqslant i(x)$. Thus $i(x)$ is a pre-fixed point of f, as required.

To apply the framework outlined above, we must produce a model which *dominates* the model $\mathcal{M}_U$. We identify a space E, an injection function $i : E \to \mathcal{F}$ and a function $\overline{f} : E \to E$ such that $\mathcal{F}_U \circ i \sqsubseteq i \circ \overline{f}$. These are defined in Appendix A.

The denotation of recursive processes can then be defined in the usual way, since the theory of local cpo's guarantees that all the relevant fixed points exist (see [MRS95] for details).

3 PTL

In this section we introduce the linear-time propositional temporal logic (PTL) [Eme90], which will be used for temporal specifications. Our version of PTL has three classes of atomic propositions. The atoms $\mathbf{P}_a$, $\mathbf{O}_a$ express the fact that a process either does an action a or offers and action a and the atom $\mathbf{D}$ expresses the fact that the process has diverged. The logic is defined in terms of a satisfaction relation $\models_t$ between observations and formulas of the logic. The expression $(s, X, d) \models_t \phi$ asserts that the formula ϕ holds on the observation (s, X, d) at time t. Observations in our model are of two types: either they are extended infinitely in time, that is end(s, X, d) is infinite; or an observation ends at some finite time t. So the assertion $(s, X, d) \models_t \phi$ can never be true for $t > \text{end}(s, X, d)$, because we read the assertion $(s, X, d) \models_t \phi$ as the observation (s, X, d) makes ϕ true at time t. If the observation supplies no information about time t then $(s, X, d) \models_t \phi$ is undefined.

The relation $\models_t$ for $t <= \text{end}(s, X, d)$ is then defined inductively as follows. For atoms:

- $(s, X, d) \models_t \mathbf{D}$ whenever $d \leqslant t$;
- $(s, X, d) \models_t \mathbf{P}_a$ whenever $(t, a) \in s$;
- $(s, X, d) \models_t \mathbf{O}_a$ whenever $(t, a) \notin X$;

So the assertion $(s, X, d) \models_t \mathbf{P}_a$ asserts that the action a is actually performed at time t. Note that we use negative information to characterize offers. We say that an observation (s, X, d) offers an a at time t if it does not refuse it.

The connectives $\wedge$ and $\neg$ are defined in the standard way:

- $(s, X, d) \models_t \phi \wedge \psi$ whenever $(s, X, d) \models_t \phi$ and $(s, X, d) \models_t \psi$;
- $(s, X, d) \models_t \neg\, \phi$ whenever $(s, X, d) \not\models_t \phi$;

The operator $\mathcal{U}$ is standard to temporal logic. Here we simply translate it into assertions on observations:

- $(s, X, d) \models_t \phi \ \mathcal{U}_{<\tau} \psi$ whenever

$$\begin{array}{r} \exists\,(\mathit{finite}) t_1 : t_1 > t \text{ and } \mid t - t_1 \mid < \tau \bullet \\ (s, X, d) \models_{t_1} \psi \text{ and} \\ \forall\, t_2 : t < t_2 < t_1 \ (s, X, d) \models_{t_2} \phi \end{array}$$

Other operators can be derived operators are defined in the usual way:

- $\phi \vee \psi$ is defined as $\neg\,(\neg\,\phi \wedge \neg\,\psi)$;
- $\phi \Rightarrow \psi$ is defined as $\psi \vee \neg\,\phi$;
- $\Diamond_{*\tau}\phi$ is defined as $\mathsf{true}\ \mathcal{U}_{*\tau}\phi$,
- $\Box\phi$ is defined as $\neg\,\Diamond(\neg\,\phi)$.

So far we have explained what it means for an observation to model a formula of PTL. A process is a collection of observations satisfying the consistency conditions from Section 2.3. This makes the definition of what it means for a process to satisfy a formula slightly subtle. Because observations in processes are prefix-closed, it is not sufficient to say that a process P satisfies a formula ϕ when all its observations do. Take for example the process

$$a \rightarrow b \rightarrow \mathit{Stop}$$

intuitively this should satisfy the formula $\mathsf{true}\,\mathcal{U}\ \mathbf{O}_b$, but for example the observation $(<>, \emptyset, \infty)$ does not eventually offer the action b. To avoid this we say that a process P satisfies a formula ϕ if for each observation in P either it satisfies it or some extension does in the $\leq_E$ ordering.

A further problem arises because refusal sets in observations are also downward closed, so given the observation

$$(<>, [0, t) \times \{b\}, \infty)$$

from the process $a \rightarrow b \rightarrow \mathit{Stop}$ the process axioms require that the observation $(<>, X, \infty)$ where

$$X = \{(t', b) \mid t' < t \text{ and } t' \text{ is rational}\}$$

hence it can appear that the process is offering an action at every irrational time point. So we refine the definition of a process satisfying a formula further so that we only consider maximal refusal sets. This leads is to the following definition.

Definition 9 We say that a process P satisfies ϕ ($P \models \phi$) if for all observations o in $\mathcal{F}_U[\![P]\!]$:

$$\exists\, o' \in \mathcal{F}_U[\![P]\!] \text{ such that } o' \geqslant_E o \wedge o' \models_0 \phi$$

This definition allows the use of an inferential style of reasoning via rules such as:

$$\frac{P \models \phi}{a \mapsto P \models \mathbf{O}_a \,\mathcal{U}_{<\infty}(\mathbf{P}_a \wedge \phi)}$$

with the side condition $(\neg\ (\phi \Rightarrow \neg\ \mathbf{P}_a))$. If ϕ is true of the process P, then when we consider the process $a \rightarrow P$ we know that the event a is offered immediately, and if it is performed then ϕ will immediately become true. Some rules are presented in Appendix B.

4 Specification and Refinement

We now come on to an important notion in this paper, the idea of specification pairs. A specification pair combines a CSP process P and a PTL formula ϕ into a single pair (P, ϕ). This pair represents the subset of observations of $\mathcal{F}_U[\![P]\!]$ which satisfy ϕ. Specification pairs give a way of mixing two development styles: process refinement as used extensively in the CSP community and the use of temporal logic for time-dependent system development.

As with the definition of $\models$ relating processes and formulae there are some similar technical considerations with incomplete observations which lead to the following definition:

Definition 10 A *specification pair* is a pair (P, ϕ) consisting of a CSP process P and a PTL formula, ϕ. It is defined in terms of observations as

$$(P, \phi) = \{o \in \mathcal{F}_U[\![P]\!] \mid \exists\, o' \geqslant_E o \,\text{with}\, o' \models \phi\}$$

It is important to realize that although the pair (P, ϕ) is a specification, it does not always denote a CSP process, but represents a wider notion than that of a process. It may then be unimplementable, for example:

$$(P = a \rightarrow P, \Box(\mathbf{O}_b))$$

does not denote a process, because the process P is never able to offer a an action b and therefore the process specification pair contains no observations. Such unimplementable specifications can be difficult to spot and arise because the behavioural and temporal specifications are in conflict. Thus when using specification pairs there are further proof obligations on the designer to show that the components of a pair are consistent with each other and that the specification is therefore implementable. Such proof obligations seem unavoidable, in a system which allows such a wide separation of concerns as we do here. Below we shall discuss some of the forms that such proof obligations take.

As with ordinary CSP processes, process specification pairs can refined, formally this gives:

Definition 11 (P, ϕ) is refined by (Q, ψ) (written $(P, \phi) \sqsubseteq (Q, \psi)$) if $(Q, \psi) \subseteq (P, \phi)$

To connect standard process refinement and deduction in PTL we present the following proposition (where $\psi \vdash \phi$ indicates that ϕ follows from ψ, the deduction system in [Jac90] could be used to prove this):

Proposition 1

$$\frac{P \sqsubseteq Q \quad \psi \vdash \phi}{(P, \phi) \sqsubseteq (Q, \psi)}$$

This proposition allows the two styles of development, process refinement and temporal logic deduction, to be combined in one framework.

Given a specification (P, ϕ), one way of showing that is it implementable is by proving that it meets the axioms for defining a process. In practice this would be tedious, but for various cases there simpler ways of checking implementability.

Given a process Q, if $Q \models \phi$ then it is easy to see that (Q, ϕ) is a process, since $(Q, \phi) = \mathcal{F}_U[\![Q]\!]$. This gives a simple sufficient condition to test whether a refinement of a specification is a process.

An example development would start with a specification (P, ψ) and proceed by a series of refinements:

$$(P, \psi) \sqsubseteq (P', \psi') \sqsubseteq \ldots \sqsubseteq (Q, \phi)$$

where the last specification can easily be shown to be an implementable specification by showing $Q \models \phi$. While Proposition 1 is powerful and useful it provides no mechanism for moving information from a formula ϕ in (P, ϕ) to the process P. We mention one proof rule which allows timing information to be moved across:

$$\frac{(a \rightarrow Q, \phi) \quad Q \models \psi \quad \phi \vdash (\neg\psi \; \mathcal{U}_{>\tau} \psi)}{(a \rightarrow Q, \phi) \sqsubseteq (a \rightarrow \mathit{Wait}(\tau) \; ; \; Q, \phi)}$$

Returning to the example of Section 2, if we wished to insist that every time the light was turned on, it stayed on for at least two seconds, we could write

$$(\mathit{Light} = on \rightarrow \mathit{off} \rightarrow \mathit{Light}, \Box_{<\infty}(\mathbf{P}_{on} \Rightarrow ((\neg\, \mathbf{O}_{\mathit{off}})\ \mathcal{U}\ _{\geqslant 2}\mathbf{O}_{\mathit{off}})))$$

which is refined by the above rule to

$$(\mathit{Light} = on \mapsto \mathit{Wait}(2)\ ;\ \ \mathit{off} \rightarrow \mathit{Light}, \Box_{<\infty}(\mathbf{P}_{on} \Rightarrow ((\neg\, \mathbf{O}_{\mathit{off}})\ \mathcal{U}\ _{\geqslant 2}\mathbf{O}_{\mathit{off}})))$$

and further refined to

$$(\mathit{Light} = on \mapsto \mathit{Wait}(2)\ ;\ \ \mathit{off} \mapsto \mathit{Light}, \mathsf{true})$$

However, some temporal logic formulae capture global constraints, and can only be refined into the CSP process when the entire system has been described. For example, if *Light* was only a subcomponent of a larger system, then the global constraint that the light remained on for at most two seconds:

$$\Box_{<\infty}(\mathbf{P}_{on} \Rightarrow (\mathsf{true}\ \mathcal{U}\ _{\leqslant 2}\mathbf{P}_{\mathit{off}}))$$

could only be verified once the CSP description had been completed.

5 Recursion

Consider the CSP process $\mu\, X.F(X)$. Iterations of the function $[\![F]\!] = f$ give rise to the constructable chain:

$$\bot\ \sqsubseteq\ f_1(\bot)\ \sqsubseteq\ f_2(\bot)\ \sqsubseteq\ f_3(\bot)\ \ldots \sqsubseteq\ f_i(\bot)\ \sqsubseteq \ldots$$

The lowest element of this chain, $\bot$, has every possible behaviour. If we said that the limit of a constructable chain models a formula whenever all its approximations do, then the only formula that recursive processes could model would be true.

We therefore consider only the non-divergent traces of an element of the chain.

Furthermore, the fact that observations are prefix-closed can present some problems. As an example, consider $\mu\, X.a \mapsto X$. One of the simplest things which is true of this process is that it always offers an a. Therefore we want $\mu\, X.a \mapsto X \models \Box\mathbf{O}_a$. But this does not hold initial segment of the constructible chain, because once the a actions have been performed, the process behaves chaotically. To solve this problem, we make use of the notion of pointwise-maximal observations, the definition of which is repeated here from Definition 2.

Definition 12 (s, X, d) is *pointwise-maximal* in $[\![P]\!]$ whenever

$$\forall (s, X', d') \in [\![P]\!].(s, X', d') \geqslant_E (s, X, d) \Rightarrow \begin{array}{l} (s, X', d') = (s, X, d) \\ \vee \\ \mathrm{end}(s, X', d') > \mathrm{end}(s, X, d) \end{array}$$

We must also introduce a new notion of satisfaction which, although defined for a general process, is designed to apply to recursive processes.

Definition 13 $[\![P]\!] \models\!\!\!\models_t \phi$ whenever for all pointwise-maximal, nondivergent $(s, X, \infty) \in [\![P]\!]$, $(s, X, \infty) \models_t \phi$, or some extension of (s, X, ∞) does.

The above definitions allow us a form of Scott-induction for a certain class of processes.

Definition 14 An *pointwise-maximal admissable predicate* is one which, if it holds (in the sense of $\models\!\!\!\models_t$) for each finite element of a constructable chain, also holds (again in the sense of $\models\!\!\!\models_t$) for the fixpoint of the chain. Formally, ϕ is a pointwise-maximal admissable predicate if, for all constructable chains f_i,

$$\forall\, i \bullet f_i(\perp) \models\!\!\!\models_t \phi \Rightarrow [\![\, \mu\, X.f(X)]\!] \models\!\!\!\models_t \phi$$

Definition 15 A *specification pair* is a pair (P, ϕ) consisting of a CSP process and a PTL formula. It is defined in terms of observations as

$$\begin{array}{rl} (P, \phi) = \{(s, X, d) \in [\![P]\!] \mid & \exists\, \mathrm{pointwise\!-\!maximal\ extension}(s', X', d') \bullet \\ & (s', X', d') \geqslant_E (s, X, d) \wedge \\ & (s', X', d') \models \phi \wedge \\ & \forall (s'', X'', d'') \geqslant_E (s', X', d') \mathrm{with}\ X''\ \mathrm{maximal} \\ & \quad (s'', X'', d'') \models \phi\} \end{array}$$

If ϕ is a pointwise-maximal admissable predicate, we have the following theorem.

Theorem 4

$$\frac{\forall\, n(F^n(\perp) \models\!\!\!\models_t \phi)}{(\mu\, X.F(X), \Box\neg\, \mathbf{D}) \models \phi}$$

Proof 4 Consider $(s, X, \infty) \in (\mu\, X.F(X), \Box\neg\, \mathbf{D})$, where X is a maximal refusal set.

Since ϕ is a pointwise-maximal admissable predicate, it follows immediately that $[\![\, \mu\, X.F(X)]\!] \models\!\!\!\models \phi$.

Thus if (s, X, ∞) is pointwise-maximal then it follows immediately that $(s, X, \infty) \models \phi$.

If (s, X, ∞) is not pointwise-maximal, then we know that there exist pointwise-maximal nondivergent extensions of (s, X, ∞) in $(\mu\, X.F(X), \Box\neg\, \mathbf{D})$. Since these are also in $[\![\, \mu\, X.F(X)]\!]$, there is some extension of (s, X, ∞) in $(\mu\, X.F(X), \Box\neg\, \mathbf{D})$ which models ϕ.

Therefore either $(s, X, \infty) \models \phi$, or some extension does.

Theorem 5 All predicates (excluding those which include $\mathbf{D}$) are pointwise-maximal admissable.

Proof 5 Consider $(s, X, \infty) \in [\![\, \mu\, X.F(X)]\!]$. If $\#s < \infty$ then $(s, X, \infty) \in f_i(\perp)$ for some i. If $\#s = \infty$ then we perform a structural induction.

The atomic propositions form the base cases.

case $\mathbf{P}_a$: If $\#s = \infty$ then $\mathrm{end}(s) = \infty$, and there exists a finite prefix $(s', X', \infty).(s', X', \infty) \not\models_t \mathbf{P}_a$.

case $\mathbf{O}_a$: as $\mathbf{P}_a$.

The remainder of the proof follows by a routine structural induction.

6 Conclusions, comparisons and future work

We have presented a specification framework which uses the languages of CSP and PTL, and cleanly separates the concerns of behaviour and timing, by allowing each constraint to be captured in the most suitable language. We have developed an enhanced denotational semantic model for CSP, and we have also interpreted the temporal logic PTL over this model. This has enabled us to combine the two formalisms at the semantic level in a uniform way, by the use specification pairs.

In [Jac90] the author presents a way of relating temporal logic with real-time CSP. The semantics for the temporal logic is given as a Kripke model, and to express properties of time-guarded processes the model of [Sch89] is reinterpreted as a Kripke model. The emphasis there is on the relation between the two models, and it is essentially concerned with the form of the relation $\models_t$. Whereas in this work we have developed a framework, which allows properties from both models to be combined.

In [Sch97], a theory of timewise refinement within the context of CSP is presented. This has a very similar goal to the work presented here, in that it allows the translations of specifications and proofs of correctness between timed and untimed semantic models. The timed interpretations proposed in [Sch97] initially allow more timed processes to be refinements of untimed ones, but the parallel composition operator does not in general preserve refinement. In our approach the timed interpretation is more selective, and parallel composition preserves refinement.

A dual language specification framework is proposed in [Bla94], using LOTOS and a temporal language called QTL. In that work, rather develop a common semantic model, the author presents a specialised technique for verifying a specification of a system presented both in QTL and LOTOS against it requirements (described in QTL). The work is carried out with the specific application domain of multimedia systems in mind.

Many questions remain open, while it is theoretically possible to reason about systems using the semantics the derivation of useful proof rules is of importance, these are being investigated in the context of a case study. An area of possible further research is to investigate the techniques developed in model checking [ACD93] to derive an algorithm for automatically checking that a process P meets a specification ϕ.

Acknowledgements

Thanks are due to Steve Schneider and John Derrick, for their valuable comments on earlier versions of this paper.

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[Bla94] Lynne Blair. *The Formal Specification and Verification of Distributed Multimedia Systems.* PhD thesis, Lancaster University, U.K., September 1994.

[Dav93] Jim Davies. *Specification and proof in real time CSP.* Cambridge University Press, 1993.

[Eme90] E. Allan Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier, 1990.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Jac90] D. M. Jackson. Specifying Timed Communicating Sequential Processes using Temporal Logic. Technical Report PRG-TR-5-90, Oxford University, U.K., 1990.

[MRS95] M. Mislove, A. Roscoe, and S. Schneider. Fixed points without completeness. *Theoretical Computer Science*, 138:273–314, 1995.

[Ree88] G. M. Reed. *A Uniform Mathematical Theory for Real-Time Distributed Computing.* PhD thesis, Oxford University, 1988.

[Ros88] A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall Series in Computer Science. Prentice Hall, 1988.

[Ros93] A. W. Roscoe. Unbounded Non-determinism in CSP. *Journal of Logic and Computation*, 3(2):131–172, 1993.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *Proceedings of ICALP '86*, 1986. LNCS 226.

[RR87] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings of the third workshop on the Mathematical Foundations of Programming Language Semantics*, 1987. LNCS 298.

[Sch89] S. A. Schneider. *Correctness and communication in real-time systems.* PhD thesis, Oxford University, 1989.

[Sch92] S. A. Schneider. Unbounded nondeterminism for real-time processes. Technical Report TR 13–92, Oxford University, 1992.

[Sch97] S A. Schneider. Timewise refinement for communicating processes. *Science of Computer Programming*, 28:43–90, 1997.

A The predeterministic space

The space $\mathcal{O}_P$ of predeterministic processes consists of all processes which satisy axioms 1-5 and are predeterministic. The function $\mathcal{F}_P$ over the space of predeterministic processes *dominates* the function $\mathcal{F}_U$. For every process P which may be constructed using the CSP operators, the corresponding process $\overline{P}$ constructed using the dominating operators will always dominate the process P. $\mathcal{F}_U[\![P]\!] \sqsubseteq \mathcal{F}_P[\![\overline{P}]\!]$.

The function $\mathcal{F}_P$ is defined as follows:

$$\mathcal{F}_P[\![\overline{\bot}]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![Stop]\!] = \{(\langle\rangle, X, \infty) \mid X \in RSET\}$$

$$\mathcal{F}_P[\![\overline{Stop}]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![Stop]\!] = \{(\langle\rangle, X, \infty) \mid X \in RSET\}$$

$$\mathcal{F}_P[\![\overline{Wait(d)}]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![\ Wait(d)]\!]$$

$$\begin{aligned}\mathcal{F}_P[\![P \overline{|[\,A\,]|}\, Q]\!] = \{(s, X, d) \mid\ & \exists\, s_P, X_P, d_P, s_Q, X_Q, d_Q \bullet \\ & (s_P, X_P, d_P) \in \mathcal{F}_U[\![P]\!] \wedge \\ & (s_Q, X_Q, d_Q) \in \mathcal{F}_U[\![P]\!] \wedge \\ & s \in s_P \overline{|[\,A\,]|}\, s_Q \wedge \\ & X = X_P \overline{|[\,A\,]|}\, X_Q \wedge \\ & d = \min\{d_P, d_Q\}\}\end{aligned}$$

where $s_P \overline{|[\,A\,]|}\, s_Q$ is defined recursively as $(a \in A, b, c \notin A)$

$$\begin{aligned}\langle\rangle \overline{|[\,A\,]|}\, \langle\rangle &= \{\langle\rangle\} \\ s_P \overline{|[\,A\,]|}\, \langle\rangle &= \{s_P\} \\ \langle\rangle \overline{|[\,A\,]|}\, s_Q &= \{s_Q\} \\ (t, a) \frown s_P \overline{|[\,A\,]|}\, (t, a) \frown s_Q &= \{(t, a) \frown (s_P \overline{|[\,A\,]|}\, s_Q) \\ (t_1, b) \frown s_P \overline{|[\,A\,]|}\, (t_2, c) \frown s_Q &= \{(t_1, b) \frown (s_P \overline{|[\,A\,]|}\, (t_2, c) \frown s_Q)\}, \text{ if } t_1 < t_2 \\ &= \{(t_2, c) \frown ((t_1, b) \frown s_P \overline{|[\,A\,]|}\, s_Q)\}, \text{ if } t_2 < t_1 \\ &= \{(t_1, b) \frown (s_P \overline{|[\,A\,]|}\, (t_2, c) \frown s_Q)\}\end{aligned}$$

and

$$X_P \overline{|[\,A\,]|}\, X_Q = X_P \,|[\,A\,]|\, X_Q$$

External choice

$$
\begin{aligned}
\mathcal{F}_P[\![\mathcal{P}\,\overline{\Box}\,Q]\!] = {} & \{(s,X,d) \mid (s,X,d) \in \mathcal{F}_P[\![P]\!] \wedge \\
& \qquad\qquad (\langle\rangle, X \uparrow [0, \mathrm{begin}\, s), \infty) \in \mathcal{F}_P[\![Q]\!]\} \\
& \cup \\
& \{(s,X,d) \mid (s,X,d) \in \mathcal{F}_P[\![Q]\!] \wedge \\
& \qquad\qquad (\langle\rangle, X \uparrow [0, \mathrm{begin}\, s] \cup s \uparrow [t,t], \infty) \in \mathcal{F}_P[\![P]\!]\}
\end{aligned}
$$

Dominated internal choice is easily defined

$$\mathcal{F}_P[\![P\,\overline{\sqcap}\,Q]\!] \mathrel{\widehat{=}} \mathcal{F}_P[\![P]\!]$$

The immediate prefix choice introduces no nondeterminism

$$\mathcal{F}_P[\![a \overline{\Rightarrow} P]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![a \mapsto P]\!]$$

The nondeterminsitic prefix operator is dominated by the immediate prefix operator.

$$\mathcal{F}_P[\![a \overline{\rightrightarrows} P]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![a \mapsto P]\!]$$

Skip introduces no nondeterminism

$$\mathcal{F}_P[\![\overline{\mathit{Skip}}]\!] \mathrel{\widehat{=}} \mathcal{F}_U[\![\mathit{Skip}]\!]$$

The sequential composition operator may introduce nondeterminism at the point of transfer of control. For example, if a single copy of the action a is observed, then it is nodeterministic whether or not control has passed in the process

$$
\begin{aligned}
P = {} & (a \rightarrow \mathit{Skip} \\
& \Box \\
& \mathit{Skip})\ ;\ \ a \rightarrow \mathit{Stop}
\end{aligned}
$$

To resolve this, we give the action ✓ precedence over all other actions.

$$
\begin{aligned}
\mathcal{F}_P[\![P\ ;\ Q]\!] = \{(s,X,d) \mid\ & (s, X \cup [0, \min\{\mathrm{end}(s,X), d\} \times \{\checkmark\}, d) \in \mathcal{F}_P[\![P]\!] \wedge \\
& \checkmark \notin \alpha(s) \wedge \\
& \forall\, u, v \bullet s = u \frown v \Rightarrow \\
& \qquad \neg\ \exists\, t_\checkmark \bullet (u \frown \langle (t_\checkmark, \checkmark)\rangle, X \uparrow [0, t_\checkmark), \infty \in \mathcal{F}_U[\![P]\!]\} \\
\cup & \\
\{(s,X,d) \mid\ & \exists\, s_P, X_P, s_Q, X_Q, d_Q, t_\checkmark \bullet \\
& (s_P \frown \langle (t_\checkmark, \checkmark)\rangle, X_P \cup [0, t_\checkmark) \times \{\checkmark\}, \infty) \in \mathcal{F}_U[\![P]\!] \wedge \\
& \forall\, u, v \bullet s = u \frown v \Rightarrow \\
& \qquad \neg\ \exists\, t_\checkmark \bullet (u \frown \langle (t_\checkmark, \checkmark)\rangle, X \uparrow [0, t_\checkmark), \infty \in \mathcal{F}_U[\![P]\!] \wedge \\
& (s_Q, X_Q, d_Q) \mathcal{F}_U[\![Q]\!] \wedge \\
& s = s_P \frown (s_Q + t_\checkmark) \wedge \\
& X = X_P \cup (X_Q + t_\checkmark) \wedge \\
& d = d_Q + t_\checkmark\}
\end{aligned}
$$

The timeout operator may introduce nondeterminism at the point of transfer of control. If both processes can perform the same action at time t, then the decision as to which process survives is nondeterministic. We resolve this by insisting that at time t, the second process may only perform events which the first process is incapable of performing.

$$
\begin{aligned}
\mathcal{F}_P[\![P \overline{\triangleright\{t\}} Q]\!] = \{(s,X,d) \mid\ & (s,X,d) \in \mathcal{F}_P[\![P]\!] \wedge \cup \\
& \min\{d, \mathrm{begin}\, s\} \leqslant t\} \\
\{(s,X,d) \mid\ & \exists\, s_Q, X_Q, d_Q \bullet \\
& (s_Q, X_Q, d_Q) \in \mathcal{F}_P[\![Q]\!] \wedge \\
& (\langle\rangle, X \uparrow [0,t] \cup \{s_Q \uparrow [t,t]\}, \infty) \in \mathcal{F}_P[\![P]\!] \wedge \\
& s = s_Q + t \wedge \\
& X - t = X_Q \wedge \\
& d = d_Q + t\}
\end{aligned}
$$

The hiding operator may introduce nondeterminism into a process, since different observations may become identical after hiding. We resolve this using a choice function *choose*, which provides a unique way of choosing a single action from any set of actions.

$$
\begin{aligned}
\mathcal{F}_P[\![P \overline{\setminus} A]\!] = \{(s,X,d) \mid\ & \exists\, s_P, X_P \bullet \\
& (s_P, X_P, d) \in \mathcal{F}_P[\![P]\!] \wedge \\
& s = s_P \setminus A \wedge \\
& X_P = X \cup [0, \min\{\mathrm{end}(s,X), d\}) \times A \wedge \\
& acc_{P,a}(s_P)\}
\end{aligned}
$$

$$
\begin{aligned}
acc_{P,A}(s_P) = \forall\, u, w, t, a \bullet\ & s = u \frown (t, a) \frown w \Rightarrow \\
& s_{P,A}(u)(t) \neq \{\} \wedge \\
& a = choose(s_{P,A}(u)(t)) \\
& \vee \\
& s_{P,A}(u)(t) = \{\}
\end{aligned}
$$

$S_{P,A}(u)(t)$ is the set of events from A open to the process P at time t after the trace u.

$$\frac{a \in \Sigma \cup \{\checkmark\}}{\mathit{Stop} \models \Box(\neg \mathbf{O}_a)} \qquad \frac{P \models \phi \quad Q \models \psi}{P \sqcap Q \models \phi \vee \psi}$$

$$\frac{}{\bot \models \mathsf{true}} \qquad \frac{P \models \phi \quad Q \models \phi}{P \,\Box\, Q \models \phi}$$

$$\frac{}{\mathit{Skip} \models \mathbf{O}_\checkmark \,\mathcal{U}_{<\infty} \mathbf{P}_\checkmark} \qquad \frac{P \models \phi \quad Q \models \psi \quad P \models \mathsf{true}\ \mathcal{U}_{<\tau}\, \mathbf{O}_\checkmark}{P\ ;\ Q \models \phi\ \mathcal{U}_{<\tau}\, \psi}$$

provided ϕ does not contain $\mathbf{P}_\checkmark$ or $\mathbf{O}_\checkmark$ as subformulae.

$$\frac{P \models \phi \quad Q \models \psi}{P \rhd\{t\}\ Q \models \phi\ \mathcal{U}_{<t}\, \psi} \qquad \frac{P \models \phi}{P \setminus A \models \phi \wedge \Box_{<\infty}(\bigwedge_{a \in A} \neg \mathbf{O}_a) \wedge \Box_{<\infty}(\bigwedge_{a \in A} \neg \mathbf{P}_a)}$$

provided ϕ contains no subterms $\mathcal{U}_{*t'}$ with $t' > t$.

provided $\mathbf{O}_a$ and $\mathbf{P}_a$ are not subformulae of ϕ, for all $a \in A$.

Table 1: Rules for combining processes and formulae

B Rules for finite processes

In this section present a set of rules for finite processes (processes which are constructed without recursion) which connect the processes constructors with the rules of the logic. The rules which are routine to work out are in table 1.

The immediate form of the event prefix operator has the rule

$$\frac{P \models \phi}{a \mapsto P \models \mathbf{O}_a\ \mathcal{U}_{<\infty}(\mathbf{P}_a \wedge \phi)}$$

with the side condition $(\neg\ (\phi \Rightarrow \neg\ \mathbf{P}_a))$. If ϕ is true of the process P, then when we consider the process $a \rightarrow P$ we know that the event a is offered immediately, and if it is performed then ϕ will immediately become true.

Unlike the process $a \mapsto P$, the process $a \rightarrow P$ introduces two arbitrary delays, one before and one after the occurrence of the event a. So the rules becomes

$$\frac{P \models \phi}{a \rightarrow P \models \mathsf{true}\ \mathcal{U}_{<\infty}(\mathbf{O}_a\ \mathcal{U}_{<\infty}(\mathbf{P}_a \wedge (\mathsf{true}\ \mathcal{U}_{<\infty}\phi)))}$$

with the same side condition: $(\neg\,(\phi \Rightarrow \neg\,\mathbf{P}_a))$.

The rule for parallel is more complicated. A first attempt, considering only the empty synchronisation set, would be something like: (where $P \mid\mid\mid Q = P \mid[\emptyset]\mid Q$)

$$\frac{P \models \phi \wedge Q \models \psi}{P \mid\mid\mid Q \models \phi \wedge \psi}$$

but we can find a simple counter example. Consider the processes $b \mapsto Stop$ and $c \mapsto Stop$. It is easy to see that $b \mapsto Stop \models \Box(\neg\,\mathbf{O}_c)$ and $c \mapsto Stop \models \mathsf{true}$, but it is equally clear that $a \mapsto Stop \mid[\emptyset]\mid c \mapsto Stop \not\models \Box(\neg\mathbf{O}_c)$. The difficulty arises because ϕ can assert negative information about Q and therefore to state the rule properly we have to characterise negative information. Thus we define two operators 'neg' and 'pos' on formulae which capture positive and negative assertions about events. These operators are defined by mutual recursion:

$\mathrm{pos}(\mathsf{true}) = \mathrm{pos}(\mathsf{false}) = \emptyset$	$\mathrm{neg}(\mathsf{true}) = \mathrm{neg}(\mathsf{false}) = \emptyset$
$\mathrm{pos}(\mathbf{O}_a) = \mathrm{pos}(\mathbf{P}_a) = \{a\}$	$\mathrm{neg}(\mathbf{O}_a) = \mathrm{neg}(\mathbf{P}_a) = \emptyset$
$\mathrm{pos}(\phi \wedge \psi) = \mathrm{pos}(\phi \vee \psi) = \mathrm{pos}(\phi) \cup \mathrm{pos}(\psi)$	$\mathrm{neg}(\phi \wedge \psi) = \mathrm{neg}(\phi \vee \psi) = \mathrm{neg}(\phi) \cup \mathrm{neg}(\psi)$
$\mathrm{pos}(\phi\ \mathcal{U}\ \psi) = \mathrm{pos}(\phi\ \mathcal{S}\ \psi) = \mathrm{pos}(\phi) \cup \mathrm{pos}(\psi)$	$\mathrm{neg}(\phi\ \mathcal{U}\ \psi) = \mathrm{neg}(\phi\ \mathcal{S}\ \psi) = \mathrm{neg}(\phi) \cup \mathrm{neg}(\psi)$
$\mathrm{neg}(\neg\,(\phi)) = \mathrm{pos}(\phi)$	$\mathrm{pos}(\neg\,(\phi)) = \mathrm{neg}(\phi)$

We add the following two side conditions to the rule:

$$\mathrm{neg}(\phi) \cap (\mathrm{neg}(\psi) \cup \mathrm{pos}(\psi)) = \emptyset \qquad \mathrm{neg}(\psi) \cap (\mathrm{neg}(\phi) \cup \mathrm{pos}(\phi)) = \emptyset$$

The formula ϕ must contain no negative information about any event in ψ, unless that event is also in the synchronisation set, and similarly for ψ.

The rules presented above are not complete in two senses. First, we have not presented rules for all of the CSP operators. Second, the rules differ from the derivation rules found in [Dav93] where the rules are complete in the sense they completely characterise the behaviours of CSP operators in logical terms. But these rules differ from ours in using the full power of first order logic, as opposed to PTL in this paper. Although it is not possible to make the rules complete in the sense [Dav93], using PTL simplifies reasoning about specifications. Furthermore, it seems that the rules could be made complete using first order temporal logic as opposed to PTL and this is being investigated.