A New CSP Operator for Optional Parallelism

Markus Roggenbach
Swansea University
Swansea
United Kingdom
m.roggenbach@swan.ac.uk

Stefan Gruner, Derrick G. Kourie,
Tinus Strauss, and Bruce W. Watson
University of Pretoria
Pretoria, South Africa
{sgruner, dkourie, tstrauss, watson}@cs.up.ac.za

Abstract

We introduce a new CSP operator for modeling scenarios characterised by partial or optional parallelism. We provide examples of such scenarios and sketch the semantics of our operator. Relevant properties are proven.

1. Motivation

CSP was introduced more than 20 years ago [4, 1, 5] and remains a popular means of specifying communicating sequential processes. A fundamental assumption is that, under parallelism, *all* processes engaging in an event have to do so jointly [10, 9]. In particular, the generalized parallel operator in indexed form, $\|_X^n{}_{i=1}\mathcal{L}_i$, requires that if one of the processes $\mathcal{L}_i, i = 1 \ldots n$, is ready to engage in an event in X, then it cannot proceed until *all* the other processes are ready to engage in that same event.

However, circumstances often arise where this requirement constrains the expressive capacity of CSP from a *practical* modeling perspective. For example, suppose that the $\mathcal{L}_i$ processes above represent "listener" or "reader" processes, using a channel c to acquire information in the alphabet $X \subseteq \alpha(c)$ from some other "announcer" process, $\mathcal{A}$.

In practice, at any given moment a subset of listener processes may not be ready to act, when the rest are. Moreover, the context often requires that processing progresses, rather than waits for *all* processes to reach a ready state. Wireless sensor networks [2] are but one of many possible contexts. Here, the announcer could be a sensor node wanting to pass on information to all live surrounding nodes, bypassing those which might have become inactive because their power has run out.

The CSP specification $\mathcal{A} \|_X (\|_X^n{}_{i=1}\mathcal{L}_i)$ does not adequately model this scenario. It insists that all processes have to engage *jointly* in events in X. In this example, *all* the listener processes will have to be in the form $c?x \rightarrow \mathcal{L}_i'$, *and* the announcing process will have to be in the form $c!x \rightarrow \mathcal{A}'$ before any interaction can occur. This obligation to act jointly (e.g. listen or read) characterises *all* existing CSP parallel operators, whether synchronized, alphabetized or general.

In some contexts, using the *interleaving* operator may ameliorate the problem sketched above. For example, $\mathcal{A} \|_X (|||_{i=1}^n \mathcal{L}_i)$ means that when *any* $\mathcal{L}_i$ is in the form $c?x \rightarrow \mathcal{L}_i'$ and $\mathcal{A}$ is in the form $c!x \rightarrow \mathcal{L}'$, then an interaction can take place. At best, then, such interleaving leads to an approximate specification, since only one of possibly several ready $\mathcal{L}_i$ processes is non-deterministically chosen for interaction with $\mathcal{A}$. (Note, however, that this form could express a supervisor-worker pattern: supervisor $\mathcal{A}$ successively communicates a sequence of independent subtasks to any available worker $\mathcal{L}_i$.)

An alternative operator for modelling scenarios similar to the announcer-listener pattern or the reader part of the reader-writer pattern would evidently be of pragmatic value. It will be seen that if our new operator conjoins one or more processes (e.g. listeners or readers), then *any one or more* of these processes may synchronize with their environment. Since, synchronization is neither mandatory over all conjoined processes (as in the case of general parallelism), nor constrained to one process only (as in the case of interleaving), we call it the *optional parallel* operator. It leads quite naturally to specifications of scenarios under consideration here[1].

2. Related Work

It will be seen below that the optional parallel operator can be used to model broadcasting. To date, the need for broadcasting been addressed by devising new process algebras.

[1] It is well-known that very limited subsets of CSP operators are sufficiently expressive to denote all elements of a semantic domain—e.g. in the traces domain $\{\rightarrow, \sqcap, STOP\}$ suffices [9]. The introduction of additional operators is therefore to serve a *practical* purpose. It is emphatically on pragmatic grounds, therefore, that we argue for a new operator.

For example, Prasad [8] used CCS [7] as a basis for his CBS which was used to model Ethernet-like communication. A further development is the $b\pi$-calculus by [3] where broadcasting is the basic communication primitive.

Kramer and Magee [6] give an announcer-listener model in FSP—a variant of CCS. An announcer process announces events to an event manager process, which disseminates them to listener processes. A listener process has to register with the event manager in order to receive a certain event type from it. If more than one listener needs to be notified of an event, a separate synchronizing event is required for each listener process. In relying on an event manager to do initial registering of listeners to event types, and to dispense sequentially messages to listeners, this FSP specification of the announcer-listener problem anticipates the well-known Java implementation strategy. The essence of the scenario can be encapsulated elegantly at a higher level of abstraction using the optional parallel operator.

Some theoretical work by Taubner [11] shows that a family of "parallel" operators could be elaborated. Preliminary work suggests that it is possible to express our new operator in his abstract programming language. The details of exactly how is the topic of ongoing work.

3. Theory

This section begins by showing how the step law of the general parallel operator can be modified to become a step law for the operator that we use for optional parallelism. The operator's semantics in terms of the CSP trace model is then given, followed by an indication of the essential operator properties, as well as the operator's failures and divergences characterisation.

3.1. Step Law

Let $\mathcal{R}_1 = ?x : A_1 \rightarrow \mathcal{R}_1'$ and $\mathcal{R}_2 = ?x : A_2 \rightarrow \mathcal{R}_2'$. Noting that the initial events of $\mathcal{R}_1 \,{}^{\parallel}_{X}\, \mathcal{R}_2$ have to be in $C = (X \cap A_1 \cap A_2) \cup (A_1 \backslash X) \cup (A_2 \backslash X)$, Roscoe [9, §2.4] provides a step law for the general parallel operator. The $\,{}^{\parallel}_{X}\,$-step law can be expressed as the *external* choice of four different processes, as follows:

$$
\begin{aligned}
&\mathcal{R}_1 \,{}^{\parallel}_{X}\, \mathcal{R}_2 = \\
&\quad (?x : X \cap A_1 \cap A_2 \rightarrow (\mathcal{R}_1' \,{}^{\parallel}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : (A_1 \cap A_2) \backslash X \rightarrow (\mathcal{R}_1' \,{}^{\parallel}_{X}\, \mathcal{R}_2) \sqcap (\mathcal{R}_1 \,{}^{\parallel}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : A_1 \backslash (X \cup A_2) \rightarrow (\mathcal{R}_1' \,{}^{\parallel}_{X}\, \mathcal{R}_2)) \\
&\quad \Box\ (?x : A_2 \backslash (X \cup A_1) \rightarrow (\mathcal{R}_1 \,{}^{\parallel}_{X}\, \mathcal{R}_2'))
\end{aligned}
$$

Note that the $\,{}^{\parallel}_{X}\,$-step law does *not* allow for any progress when the environment offers nothing other than some $x \in ((X \cap A_1) \backslash A_2) \cup ((X \cap A_2) \backslash A_1)$. Thus, if process $\mathcal{P} = (x \rightarrow \mathcal{P}')$ then $\mathcal{P} \,{}^{\parallel}_{X}\, (\mathcal{R}_1 \,{}^{\parallel}_{X}\, \mathcal{R}_2)$ will *deadlock*. This corresponds identically to the scenario where only one of two reader processes, $\mathcal{R}_1$ or $\mathcal{R}_2$, is ready to read from process $\mathcal{P}$, which is in turn ready to allow such a reading action.

Our optional parallel operator, denoted by $\,{}^{\Uparrow}_{X}\,$, lifts this restriction. Its step law needs to indicate what is to happen when the environment offers $x \in ((X \cap A_1) \backslash A_2)$ or $x \in ((X \cap A_2) \backslash A_1)$. Evidently, in the first case an interaction between the environment and $\mathcal{R}_1$ should occur, and in the second case, an interaction between the environment and $\mathcal{R}_2$. The $\,{}^{\Uparrow}_{X}\,$-step law is therefore:

$$
\begin{aligned}
&\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2 = \\
&\quad (?x : X \cap A_1 \cap A_2 \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : (A_1 \cap A_2) \backslash X \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2) \sqcap (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : A_1 \backslash (X \cup A_2) \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2)) \\
&\quad \Box\ (?x : A_2 \backslash (X \cup A_1) \rightarrow (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : (X \cap A_1) \backslash A_2 \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2)) \\
&\quad \Box\ (?x : (X \cap A_2) \backslash A_1 \rightarrow (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2'))
\end{aligned}
$$

This can be simplified to:

$$
\begin{aligned}
&\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2 = \\
&\quad (?x : X \cap A_1 \cap A_2 \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : (A_1 \cap A_2) \backslash X \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2) \sqcap (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')) \\
&\quad \Box\ (?x : A_1 \backslash A_2 \rightarrow (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2)) \\
&\quad \Box\ (?x : A_2 \backslash A_1 \rightarrow (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2'))
\end{aligned}
$$

Now, for $\mathcal{P} = (x \rightarrow \mathcal{P}')$, the process $\mathcal{P} \,{}^{\parallel}_{X}\, (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2)$ will lead to the desired reader behaviour: the process evolves into $\mathcal{P}' \,{}^{\parallel}_{X}\, (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2)$ or $\mathcal{P}' \,{}^{\parallel}_{X}\, (\mathcal{R}_1 \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')$, depending on whether $x \in (X \cap A_1) \backslash A_2$ or $x \in (X \cap A_2) \backslash A_1$ respectively. Of course, if $x \in (X \cap A_1 \cap A_2)$ then the process evolves into $\mathcal{P}' \,{}^{\parallel}_{X}\, (\mathcal{R}_1' \,{}^{\Uparrow}_{X}\, \mathcal{R}_2')$.

Note that the operator can model a broadcasting system in the following sense. Suppose that $\mathcal{P}$ is to broadcast messages to all ready $\mathcal{R}_i$ processes, but also to progress if no $\mathcal{R}_i$ is ready. Such a scenario is easily modelled by including a RUN_X as an additional $\mathcal{R}_i$ process, where $RUN_X = ?x : X \rightarrow RUN_X$. The fact that such a system embeds a divergence, is precisely a characteristic of a system that allows such unbridled broadcasting.

Considered from an operational semantics perspective, the *firing rules* for optional parallelism can be defined analogously to the ones given by [9, §7.3]. In fact, all Roscoe's firing rules for general parallelism apply *pari passu* to optional parallelism, with two additional rules as follows:

$$
\frac{\mathcal{P} \xrightarrow{a} \mathcal{P}', \mathit{not}(\mathcal{Q} \xrightarrow{a} \mathcal{Q}')}{\mathcal{P} \,{}^{\Uparrow}_{X}\, \mathcal{Q} \xrightarrow{a} \mathcal{P}' \,{}^{\Uparrow}_{X}\, \mathcal{Q}} \quad (a \in X)
$$

and analogously for its symmetrical counterpart indicating when Q fires. This is because, in both cases, if one of the processes, whether P or Q, is switched on and the other not, then $P \overset{\Uparrow}{}_X Q$ fires, whether or not the action in question is in X or not; (compare especially the case in [9] where $a \notin X$).

3.2. Semantics (Trace Model $\mathcal{T}$)

To specify the trace semantics, $\overset{\Uparrow}{}_X$ is used below as a binary function that maps two traces onto a set of traces. The mapping is defined by the relationships below. In these relationships, s or $s\hat{}\langle\surd\rangle$ and t or $t\hat{}\langle\surd\rangle$ represent the elements of $Traces(\mathcal{R}_1)$ and $Traces(\mathcal{R}_2)$ respectively, $\langle\rangle$ denotes the empty trace, and $\surd$ is the successful termination event that is per definition not in the alphabet, Σ. For $s,t \in \Sigma^*$ we define:

$$
\begin{array}{lcl}
s \overset{\Uparrow}{}_X t & = & t \overset{\Uparrow}{}_X s \\
\langle\rangle \overset{\Uparrow}{}_X t & = & \{t\} \\
\langle a\rangle\hat{}s \overset{\Uparrow}{}_X \langle b\rangle\hat{}t & = & \begin{cases} \{\langle a\rangle\hat{}u \mid u \in s \overset{\Uparrow}{}_X t\} & \text{if } a = b \wedge a \in X \\ \{\langle a\rangle\hat{}u \mid u \in s \overset{\Uparrow}{}_X \langle b\rangle\hat{}t\} \cup & \\ \{\langle b\rangle\hat{}u \mid u \in \langle a\rangle\hat{}s \overset{\Uparrow}{}_X t\} & \text{if } a \neq b \vee a \notin X \end{cases} \\
s\hat{}\langle\surd\rangle \overset{\Uparrow}{}_X t & = & \emptyset \\
s \overset{\Uparrow}{}_X t\hat{}\langle\surd\rangle & = & \emptyset \\
s\hat{}\langle\surd\rangle \overset{\Uparrow}{}_X t\hat{}\langle\surd\rangle & = & \{u\hat{}\langle\surd\rangle \mid u \in s \overset{\Uparrow}{}_X t\}
\end{array}
$$

The set of traces resulting from $\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2$ is the union of all trace sets yielded by the binary trace function, $\overset{\Uparrow}{}_X$, when applied to all possible traces from $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively:

$$
\begin{array}{l}
Traces(\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2) = \\
\quad \bigcup \{s \overset{\Uparrow}{}_X t \mid s \in Traces(\mathcal{R}_1) \wedge t \in Traces(\mathcal{R}_2)\}
\end{array}
$$

The last three relationships describing the binary trace function determine the behaviour of the optional parallel operator when applied to two terminating processes: the composite process only terminates when both participating processes jointly terminate. In the trace model for $\overset{\|}{}_X$ given in [9] that corresponds to the foregoing, only two of the last three relationships of the corresponding binary trace function are given: If s and t are typed as belonging to Σ^*, as is the case here, then the three relationships are needed for completeness.

It can easily be shown that the optional parallel operator conforms to the following:

Proposition 1. *If $Traces(\mathcal{R}_1)$ and $Traces(\mathcal{R}_2)$ are non-empty and prefix-closed, then $Traces(\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2)$ is also non-empty and prefix-closed.*

Thus, $\overset{\Uparrow}{}_X$ is well behaved in the $\mathcal{T}$ model, since it specifies a type-correct process, provided its operands are type-correct.

3.3. Operator Properties

The optional parallel operator possesses the following properties over the traces model, $\mathcal{T}$.

Theorem 1. *$\overset{\Uparrow}{}_X$ is continuous over the traces model.*

Proof. In order to define the optional parallel operator as the lifting of a (family) of trace relations as discussed in [9, §8.2], let $[\overset{\Uparrow}{}_X]_{1,2} = \{(s,t,u) \mid u \in s \overset{\Uparrow}{}_X t\}$, where $s \overset{\Uparrow}{}_X t$ is defined as above. Then from theorem 8.2.1 in [9] it follows that $\overset{\Uparrow}{}_X$ is continuous over the traces model. □

Theorem 2. *$\overset{\Uparrow}{}_X$ is symmetric and associative.*

As a result the $\overset{\Uparrow}{}_X$ operator may also be used in an indexed fashion over a finite set of processes: $\overset{\Uparrow}{}_X \overset{n}{}_{i=1} \mathcal{R}_i$.

Theorem 3. *The following additional laws hold:*

1. $\mathcal{R} \overset{\Uparrow}{}_X \mathcal{R} = \mathcal{R}$ *if* $\mathcal{R} = (?x : C \to \mathcal{R}') \wedge C \subseteq X$
2. $\mathcal{R}_1 \overset{\Uparrow}{}_{\{\}} \mathcal{R}_2 = \mathcal{R}_1 \,|||\, \mathcal{R}_2$
3. $STOP \overset{\Uparrow}{}_X SKIP = STOP$
4. $(?x : C \to \mathcal{R}) \overset{\Uparrow}{}_X STOP = ?x : C \to \mathcal{R}$
5. $(?x : C \to \mathcal{R}) \overset{\Uparrow}{}_X SKIP = ?x : C \to (\mathcal{R} \overset{\Uparrow}{}_X SKIP)$
6. $\mathcal{R}_1 \overset{\Uparrow}{}_X (\mathcal{R}_2 \,\Box\, \mathcal{R}_3) = (\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2) \,\Box\, (\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_3)$
7. $\mathcal{R}_1 \overset{\Uparrow}{}_X (\mathcal{R}_2 \sqcap \mathcal{R}_3) = (\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2) \sqcap (\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_3)$
8. $(\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2) \setminus Y = (\mathcal{R}_1 \setminus Y) \overset{\Uparrow}{}_X (\mathcal{R}_2 \setminus Y)$, *if* $X \cap Y = \emptyset$
9. $f[\mathcal{R}_1 \overset{\Uparrow}{}_X \mathcal{R}_2] = f[\mathcal{R}_1] \overset{\Uparrow}{}_{f(X)} f[\mathcal{R}_2]$, *if f is bijective.*

We provide below the proof of (3.5). Since proofs of the remaining laws are straightforward, we omit them for brevity.

Proof. (Of 3.5)

$$
\begin{array}{l}
Traces((?x : C \to P) \overset{\Uparrow}{}_X SKIP) \\
\quad = \bigcup \{s \overset{\Uparrow}{}_X t \mid s \in Traces(?x : C \to P), t \in Traces(SKIP)\} \\
\quad = \bigcup \{s \overset{\Uparrow}{}_X t \mid s \in \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid a \in C, s \in Traces(P[a/x])\}, \\
\qquad t \in \{\langle\rangle, \langle\surd\rangle\}\} \\
\quad = \langle\rangle \overset{\Uparrow}{}_X \langle\rangle \cup \{\langle a\rangle\hat{}s \overset{\Uparrow}{}_X \langle\rangle \mid a \in C, s \in Traces(P[a/x])\} \cup \\
\qquad \langle\rangle \overset{\Uparrow}{}_X \langle\surd\rangle \cup \{\langle a\rangle\hat{}s \overset{\Uparrow}{}_X \langle\surd\rangle \mid a \in C, s \in Traces(P[a/x])\} \\
\quad = \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \overset{\Uparrow}{}_X \langle\rangle \mid a \in C, s \in Traces(P[a/x])\} \cup \\
\qquad \emptyset \cup \{\langle a\rangle\hat{}s \overset{\Uparrow}{}_X \langle\surd\rangle \mid a \in C, s \in Traces(P[a/x])\} \\
\quad = \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid a \in C, s \in \\
\qquad \bigcup \{s_1 \overset{\Uparrow}{}_X s_2 \mid s_1 \in Traces(P[x/a]), s_2 \in \{\langle\rangle, \langle\surd\rangle\}\}\} \\
\quad = \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid a \in C, s \in Traces(P[x/a] \overset{\Uparrow}{}_X SKIP)\} \\
\quad = \{\langle\rangle\} \cup \{\langle a\rangle\hat{}s \mid a \in C, s \in Traces((P \overset{\Uparrow}{}_X SKIP)[x/a])\} \\
\quad = Traces(?x : C \to (P \overset{\Uparrow}{}_X SKIP))
\end{array}
$$

□

In the traces model, generalized parallelism *refines* optional parallelism. i.e. $Traces(\mathcal{R}_1 \parallel_X \mathcal{R}_2) \subseteq Traces(\mathcal{R}_1 \Uparrow_X \mathcal{R}_2)$, alternatively: $(\mathcal{R}_1 \Uparrow_X \mathcal{R}_2) \sqsubseteq_{\mathcal{T}} (\mathcal{R}_1 \parallel_X \mathcal{R}_2)$. Moreover, it is easy to see that eliminating all interleaved traces from the two parallel operator trace sets yields the same trace set, i.e.:

$Traces(\mathcal{R}_1 \parallel_X \mathcal{R}_2) \backslash Traces(\mathcal{R}_1 \,|||\, \mathcal{R}_2) =$

$Traces(\mathcal{R}_1 \Uparrow_X \mathcal{R}_2) \backslash Traces(\mathcal{R}_1 \,|||\, \mathcal{R}_2)$.

However, in general, optional parallelism and interleaving are not in a refinement relationship, i.e. *neither*

$Traces(\mathcal{R}_1 \Uparrow_X \mathcal{R}_2) \subseteq Traces(\mathcal{R}_1 \,|||\, \mathcal{R}_2)$

nor vice-versa. As a counter-example, consider $\mathcal{R}_1 = \mathcal{R}_2 = a \rightarrow SKIP$. $Traces(\mathcal{R}_1 \,|||\, \mathcal{R}_2)$ consists of $\langle a, a, \checkmark \rangle$ and all its prefixes, which does not include $\langle a, \checkmark \rangle$, this being a trace of $\mathcal{R}_1 \Uparrow_{\{a\}} \mathcal{R}_2$. Furthermore, $\langle a, a, \checkmark \rangle$ is not a trace of $\mathcal{R}_1 \Uparrow_{\{a\}} \mathcal{R}_2$.

3.4. Failures and Divergences

The $\Uparrow_X$ operator is not a source of divergence, but simply reflects the divergences of its operands. In this sense, its behaviour is similar to $\parallel_X$, and its divergences can be recursively defined in a similar manner to the definition of divergences for $\parallel_X$ given in [9] (with $\mathcal{R}_{opt} = (\mathcal{R}_1 \Uparrow_X \mathcal{R}_2)$):

$$divergences(\mathcal{R}_{opt}) = \bigcup \{ s \Uparrow_X t \mid s \in divergences(\mathcal{R}_1) \vee t \in divergences(\mathcal{R}_2) \}$$

Similarly (and in contrast to general parallelism) optional parallelism does not of itself introduce failures. Rather, a failure in $\mathcal{R}_{opt}$ arises if and only if one of its traces leads to a failure in *both* $\mathcal{R}_1$ and $\mathcal{R}_2$. In this case, the failure events are limited to those which are common failure events for *both* operand processes at that stage of their respective evolution. This can be expressed in the following predicate:

$$\forall (s, S) : failures(\mathcal{R}_1) \cdot left(\forall (t, T) : failures(\mathcal{R}_2) \cdot (\forall z : s \Uparrow_X t \cdot (((z, S \cap T) \in failures(\mathcal{R}_{opt})))))$$

As a result, the strict set of failures for optional parallelism is given by:

$$failures_{\perp}(\mathcal{R}_{opt}) = failures(\mathcal{R}_{opt}) \cup \{(s, Z) \mid s \in divergences(\mathcal{R}_{opt})\}$$

While general parallelism refines optional parallelism in the traces model, this is not the case in the failures/divergences model. Letting $\mathcal{R}_{gen} = (\mathcal{R}_1 \parallel_X \mathcal{R}_2)$, note that it is indeed the case that $divergences(\mathcal{R}_{gen}) \subseteq divergences(\mathcal{R}_{opt})$. This is easily verified by noting that while every divergence in $\mathcal{R}_{gen}$ is also a divergence in $\mathcal{R}_{opt}$, the inverse does not hold. (For example, a trace of $\mathcal{R}_{gen}$ may lead to failure because the first events in the operand processes are supposed to synchronise and do not. In the case of $\mathcal{R}_{opt}$, this same trace will not lead to failure, but it could eventually lead to a divergence.) However, there is *no* set containment relationship between $failures(\mathcal{R}_{opt})$ and $failures(\mathcal{R}_{gen})$. To see this, suppose that in $\mathcal{R}_{opt}$ and $\mathcal{R}_{gen}$ the following holds: $X = \{a, b\}$, $\mathcal{R}_1 = (a \rightarrow STOP)$ and $\mathcal{R}_2 = (b \rightarrow STOP)$. In this case, $(\langle ab \rangle, X)$ is a failure of $\mathcal{R}_{opt}$ but is not a failure of $\mathcal{R}_{gen}$ (because $\langle ab \rangle$ is not even a trace of $\mathcal{R}_{gen}$); whereas $(\langle \rangle, X)$ is a failure of $\mathcal{R}_{gen}$ but is not a failure of $\mathcal{R}_{opt}$.

4. Conclusion

In this paper we have cited various scenarios where a new CSP operator to express partial or optional parallelism would appear to be most convenient. We have introduced such an operator and demonstrated some of its properties. A formal elaboration of the proofs of all these properties has not been given, due to space constraints. Future work includes the investigation of other laws and properties of the operator. Outlines of the proofs of the equivalence relationships between the various semantic descriptions (namely step rules vs. trace semantics, trace semantics vs. operational semantics) have been worked out and will be reported elsewhere. Also on the future work agenda, is to ensure that the optional parallel operator is *tool*-supported, such that practical modelling of experiments as well as automated model-checking and/or theorem-proving can become possible.

References

[1] A. Abdallah, C. Jones, and J. Sanders, editors. *Communicating Sequential Processes: The First 25 Years*, volume 3525 of *LNCS*. Springer, 2005.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Comp. Netw.*, 38:393–422, 2002.

[3] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. *Proc. 15th Internat. Symp. on Parallel and Distr. Processing*, pages 1516–1525, 2001.

[4] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[5] T. Hoare. Why ever CSP? *ENTCS*, 162:209–215, 2006.

[6] J. Magee and J. Kramer. *Concurrency: State models and Java Programs*. John Wiley, 2 edition, 2006.

[7] R. Milner. *Communication and concurrency*. Prentice Hall, 1989.

[8] K. Prasad. A calculus of broadcasting systems. *Sc. of Comp. Programming*, 25(2-3):285–327, 1995.

[9] A. W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1997.

[10] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[11] D. Taubner. *Finite representations of CCS and TCSP programs by automata and Petri nets*, volume 369 of *LNCS*. Springer-Verlag, New York, 1989.