Under consideration for publication in Formal Aspects of Computing

On the expressive power of CSP refinement

A.W. Roscoe

Abstract. We show that wide ranging classes of predicates on the failures-divergences model for CSP can be represented by refinement checks in a general form. These are predicates of a process P expressible as $F(P) \sqsubseteq G(P)$, where F and G are CSP contexts and $\sqsubseteq$ is refinement. We use ideas similar to full abstraction, but achieve a stronger property than that. Our main result is that topologically-closed predicates are precisely those representable when F and G are both uniformly continuous. We show that sub-classes of predicates such as refinement-closed and distributive ones are represented by special forms of this check.

Keywords: CSP, refinement, topology, full abstraction

1. Introduction

The purpose of this paper is to classify the types of predicates[1] which can be decided, in its standard models, of processes in the CSP process algebra using refinement checking. The main motivation of this is to determine what can be achieved using FDR [FDR], which is a refinement checker for CSP. The two main references for CSP are [Hoa85, Ros98]. Throughout this paper we will use the notation and conventions of the second of these (the author's book).

This paper revises and extends a draft which was presented at AVOCS '03 [Ros03]. The main addition is Section 5.

Both Hoare's use of P **sat** R, where R is a property of behaviours rather than sets of behaviours, and the fact that FDR is the only widely-used verification tool for CSP, have tended to concentrate the language's users on so-called *behavioural* predicates, namely ones which are judged true of a process when and only when all the process's behaviours (traces, failures, divergences) are acceptable. Every specification of the form P **sat** R can be checked by testing if P refines the *characteristic* process of R (namely the nondeterministic composition of all processes Q such that Q **sat** R). Furthermore, the most natural style of using FDR is to check that $Spec \sqsubseteq P$ for some $Spec$ representing the predicate being asserted. In the second case (FDR), process P passes just when all its behaviours are ones of $Spec$ (since refinement is defined to mean reverse

Correspondence and offprint requests to: A.W. Roscoe, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK. *email* `Bill.Roscoe@comlab.ox.ac.uk`

[1] A predicate is simply a condition that maps a process to true or false, in other words a property of processes. The reason why we use the term "predicate" in this paper is to distinguish linguistically the properties of processes we are trying to characterise, from the many other uses of the word "property" in this paper.

containment over behaviours). Of course in both cases the precise types of behaviours checked will depend on which CSP model is used.

Let us term a predicate of the form $Spec \sqsubseteq P$ (for P the process we are checking and $Spec$ fixed) a *simple* refinement check. Of course the large number of applications of behavioural predicates and that fact that functional specifications are naturally of this sort suggest that most practical predicates one will wish to prove will be behavioural.

However there are more sophisticated specifications which can not be so expressed, usually ones which judge combinations of a process's behaviours rather than individual ones. The classic example of this is the extensional notion of *determinism* from the failures/divergences model:

- P must be divergence-free, and
- if P has trace $s\hat{\ }\langle a\rangle$ then it does not have the failure $(s, \{a\})$: P cannot have the option either to accept or refuse a. (In fact, for technical reasons, this second condition implies divergence freedom.)

The first clause of this can be checked by refinement against the most nondeterministic divergence-free process $Chaos_\Sigma$, but the second cannot. A striking way of seeing this is to observe that every predicate of the form $P \sqsupseteq Spec$ is *distributive* – if P and Q satisfy it then so does $P \sqcap Q$ – but it is self evident that determinism is not distributive.

We will see later that, in common with many other such predicates, though determinism is not checkable via a simple refinement check, it is decidable using a refinement check of a more elaborate sort.

In Sections 1–4 of this paper we will concentrate mainly on a single model, namely the failures-divergences one, making two further assumptions:

- We only consider predicates which exclude divergence, which is (as remarked above) a straightforward simple refinement check.
- We use the model without the addition of ✓ (termination), and therefore no process we consider can, at the outermost level, be capable of terminating via $SKIP$.

Let us call the divergence- and ✓-free portion of the failures-divergences model $\mathcal{N}^+$. (The whole model is generally termed $\mathcal{N}$ [Ros98].) The main reason for these restrictions is that they simplify the order-theoretic properties of the model, giving us the following proposition and reducing the number of special cases in definitions and arguments. In particular it yields the following.

PROPOSITION 1.1 *Each process P in $\mathcal{N}^+$ is equivalent to the nondeterministic choice of all deterministic processes that refine it, namely $\bigsqcap\{Q \in \mathcal{N}^+ \mid Q \sqsupseteq P \wedge Q \text{ is deterministic}\}$.*

Furthermore, since this is in any case a practical requirement for FDR, we will assume that the alphabet Σ over which processes are built is finite. It also has the following mathematical advantages:

- All definable CSP operators become continuous in the standard order-theoretic sense, using the refinement order.
- Point-set topology is important in the classification of properties over partial orders like CSP models; there are various topologies over CSP models described in [Ros91, Ros92] but as established in those papers they all collapse to the same one when Σ is finite; this has numerous useful properties such as compactness.

We will, however, feel free to add finite collections of events to our alphabet for the purpose of expressing predicates.

The rest of this paper is structured as follows. In the next section we will prove a general theorem (the main one of this paper) that shows that an unexpectedly large class of predicates are all expressible as refinement checks. In the following sections we look at two important subclasses of predicate, namely *refinement-closed* ones and *distributive ones*, which each have distinctive styles of representation and which, in the author's experience, constitute most practical predicates that are not behavioural. We consider various examples of these two classes, which are formally defined:

DEFINITION 1.1 *The predicate R is* refinement-closed *if and only if $R(P)$ and $P \sqsubseteq P'$ implies $R(P')$. Namely, if P satisfies the predicate then all its refinements do.*

R is distributive *if and only if whenever S is a nonempty set of processes satisfying R, then $R(\bigsqcap S)$.*

Finally we examine what can be achieved if we allow our representations to make use of divergence. For reasons which will become apparent, we concentrate mainly on traces predicates in that section.

There is an appendix of CSP and other notation.

2. General predicates

The most general basic predicate that can be decided by refinement-checking is of the form

$$R(P) \equiv F(P) \sqsubseteq G(P) \qquad (\dagger)$$

where F and G are both CSP contexts (i.e., process expressions which may involve the process variable P). The most general predicates are then logical combinations (using $\wedge$, $\vee$, $\neg$) of these. For practical purposes we should restrict ourselves to combinations of a finite number of these basic predicates, each of which has the form that $F(P)$ and $G(P)$ are always finite-state when P is – otherwise deciding the predicate would generate an infinite amount of work for FDR. However this paper explores well beyond this practical domain.

Most of our effort will be spent on basic checks of the form in (†). It is natural to identify each predicate with the subset of $\mathcal{N}^+$ consisting of those processes satisfying it. We will often make this identification.

Conjunction and disjunction

Before we study how to represent individual predicates, it is interesting to show that one can represent both conjunctions (finite and countably infinite) and finite disjunctions using refinement checks alone. In other words, one of these logical combinations of refinement checks can be represented as a single refinement check.

Suppose first that $F_i(P) \sqsubseteq G_i(P)$ are all representations of predicates $R_i(P)$. Then if we pick new events a, b, consider the processes $F^*(P) = \bigsqcap\{nas(n,b);\ F_n(P)\}$ and $G^*(P) = \bigsqcap\{nas(n,b);\ G_n(P)\}$, where

$$\begin{array}{rcl} nas(0,c) & = & c \to SKIP \\ nas(n+1,c) & = & a \to nas(n,c) \end{array}$$

It is not too hard to see that $F^*(P) \sqsubseteq G^*(P)$ if and only if $F_i(P) \sqsubseteq G_i(P)$ for all i, since we know instantly by looking at a trace which of the $F_i(P)$ or $G_i(P)$ it relates to, unless it is a trace consisting only of a's, on which the two sides have identical behaviour. The single refinement will only hold if all the constituent ones do, and vice-versa. It is trivial to cut this down to a finite conjunction, but it cannot be extended from these uncountably infinite conjunctions to uncountably infinite ones without an uncountable alphabet, which we do not have.

For disjunction, at least over the failures model, we need to extend the alphabet further. Given the refinements $F_i(P) \sqsubseteq G_i(P)$ for $i \in \{1,2\}$, choose an injective renaming R (extending the alphabet if necessary) such that $F_2(P)[\![R]\!]$ and $G_2(P)[\![R]\!]$ have disjoint alphabets from both $F_1(P)$ and $G_1(P)$. (Note that applying an injective renaming to both sides of a refinement test never changes the outcome.) Call the resulting alphabets A_1 and A_2. The refinement

$$(Chaos_{A_1} \mathbin{|||} F_2(P)[\![R]\!]) \sqcap (F_1(P) \mathbin{|||} Chaos_{A_2}) \sqsubseteq G_1(P) \mathbin{|||} G_2(P)[\![R]\!]$$

is then equivalent to the disjunction of the two parts. If either refinement does hold, then by construction the above holds (as $Chaos_A$ is minimal amongst divergence-free processes with alphabet a subset of A). If it were to hold and neither constituent does then we can find failures[2] (s, X), (t, Y) respectively of $G_1(P)$ and $G_2(P)[\![R]\!]$ which do not (respectively again) belong to $F_1(P)$ or $F_2(P)[\![R]\!]$. The failure $(s\hat{\ }t, X \cap Y)$ is one of $G_1(P) \mathbin{|||} G_2(P)[\![R]\!]$ (where $s\hat{\ }t$ could be replaced by any interleaving of s and t) but not one of $Chaos_{A_1} \mathbin{|||} F_2(P)[\![R]\!]$ or $F_1(P) \mathbin{|||} Chaos_{A_2}$. This is because for $(s\hat{\ }t, X \cap Y)$ to be a failure of $Q \mathbin{|||} Q'$, where Q and Q' respectively have alphabets A_1 and A_2, we must have $(s, X) \in failures(Q)$ and $(t, Y) \in failures(Q')$ by the disjointness of A_1 and A_2 and our assumptions of X and Y. This gives a contradiction, proving our claim.

An alternative construction for disjunction over the traces model can be found in Section 5, as can a discussion of infinite disjunction.

---

[2] Here X and Y are ranging over subsets of the whole alphabet Σ, not just those of particular processes. Indeed we assume here – as we may – that $\Sigma \setminus A_1 \subseteq X$ and $\Sigma \setminus A_2 \subseteq Y$.

We note here (because it will be of interest later) that the constructions given here for both the left- and right-hand sides of conjunctions are distributive in P if all the F_i/G_i are. However the construction used for the right-hand side of disjunction is not distributive since it involves putting two versions of P in parallel with each other. Later we will see that this non-distributive construction for disjunction is inevitable.

Uniform continuity

Over the finite-Σ case that we are studying, most CSP-definable functions have the following property.

Definition 2.1 *A CSP-definable function F is said to be uniformly continuous (relative to a given model) if, for every length of trace k, there is another length $m(k)$ such that, for all processes P,*

$$P \downarrow m(k) = Q \downarrow m(k) \Rightarrow F(P) \downarrow k = F(Q) \downarrow k \qquad (UC)$$

where $P \downarrow n$ behaves exactly like P until n events have been performed, and then diverges (see [Ros98]). In other words the behaviour of $F(P)$ up to step k is completely determined by the behaviour of P up to step $m(k)$.

This definition is similar to, but much weaker than, those of the familiar CSP concepts of a constructive or non-destructive function – see [Ros98].

The only exceptions to (UC) for CSP-definable F and finite Σ are

- When $F(\cdot)$ may introduce divergence through hiding (i.e. $F(P)$ can diverge when P does not). This does not exclude all uses of hiding, provided the structure of $F(\cdot)$ guarantees that no consecutive infinite sequence of events will be hidden. *In fact, we will use such safe hiding often in the F's and G's we build.*
- When unbounded nondeterminism creates a choice between an infinity of processes which have, for a fixed k, arbitrarily large $m(k)$'s.

From the standpoint of mathematical analysis (UC) is closely analogous to uniform continuity with respect to the usual metric on the model

$$d(P, Q) = \inf\{2^{-k} \mid P \downarrow k = Q \downarrow k\}$$

(A function from one metric space (A, d_1) to another (B, d_2) is said to be uniformly continuous if, for all $\epsilon > 0$ there exists $\delta > 0$ such that for all x and y in A, $d_1(x, y) < \delta \Rightarrow d_2(f(x), f(y)) < \epsilon$.) See [Ros98, Ros92, Ros91] for details on metric spaces over CSP models. Appendix A of [Ros98] contains a tutorial on metric spaces from the CSP perspective. In fact the properties (specifically topological compactness[3]) of $\mathcal{N}$ with Σ finite mean uniform continuity is equivalent to ordinary metric continuity (where δ is allowed to vary with x as well as with ϵ).

Definition 2.2 *A predicate R is said to be* closed *if the set $\{P \mid R(P)\}$ is closed in the metric topology.*

So suppose F and G are both (UC) CSP contexts which cannot introduce divergence. (This is equivalent to $F(Chaos_\Sigma)$ and $G(Chaos_\Sigma)$ being divergence free.) Since any failure of the refinement (†) shows up in some finite length behaviours (i.e., if $H \downarrow n \sqsubseteq K \downarrow n$ for all n then $H \sqsubseteq K$) it follows from (UC) that the predicate R defined as in (†) satisfies

$$\neg R(P) \;\Rightarrow\; \exists\, k. \forall\, Q.(Q \downarrow k = P \downarrow k \Rightarrow \neg R(Q))$$

This is because, if $\neg R(P) \equiv F(P) \not\sqsubseteq G(P)$ then we can pick n with $F(P) \downarrow n \not\sqsubseteq G(P) \downarrow n$. Setting k to be the greater of $m_F(n)$ and $m_G(n)$ (the functions demonstrating (UC) for F and G) then gives the right-hand side of the above implication.

The above is the usual definition of the *continuity* property (for predicates) (see [Ros98]) and says that the set of processes satisfying R is closed in the metric space. (Namely, if $\langle x_i \rangle$ is a convergent sequence of processes in R, then its limit is also in R.) Indeed a predicate is closed if and only if it is continuous: these are just two ways of viewing the same phenomenon.

[3] Compactness is a strong property of a topological space which says, in a fairly strong sense, that the whole space is bounded. In a metric space, as we have here, compactness is equivalent to there not existing an infinite set D of points with no limit point, namely a point z such that for every $\epsilon > 0$ there are infinitely many points of D within ϵ of z.

Representing closed predicates

We have thus shown that all predicates representable by (†) using (UC) F and G are closed. Our main result is going to be the reverse of this: showing that *all* closed predicates are expressible using (†) using F and G which satisfy (UC). It is helpful to turn our attention from the metric space to one of the other topological views of CSP from [Ros91, Ros92], namely the δ-topology[4].

Definition 2.3 *The δ-topology on $\mathcal{N}$ is defined by specifying that a set $C \subseteq \mathcal{N}$ is closed if and only if, whenever $P \notin C$, there exist finite sets Φ of failures and Δ of divergences such that for all Q,*

$$\mathit{failures}(P) \cap \Phi = \mathit{failures}(Q) \cap \Phi \wedge \mathit{divergences}(P) \cap \Delta = \mathit{divergences}(Q) \cap \Delta \;\Rightarrow\; Q \notin C$$

(A topology is defined by its set of closed sets.)

Note that, unlike the metric one, the δ-topology is not based on lengths of traces. The reason we use the latter here is that it allows us to understand closed predicates in a rather intuitive way, as we will see. This is especially important when we come to study distributive and refinement closed predicates.

For $\mathcal{N}^+$ (where all processes are divergence free) we need only consider Φ: a predicate is closed if and only if, whenever $\neg R(P)$, there is a finite set Φ such that for any Q, if

$$\mathit{failures}(P) \cap \Phi = \mathit{failures}(Q) \cap \Phi$$

then $\neg R(Q)$.

For example, the predicate defined $R(P) \equiv Q \sqsubseteq P$ (for any fixed Q) is closed because $\neg R(P)$ implies that there is $f \in \mathit{failures}(P) \setminus \mathit{failures}(Q)$. Evidently if $\mathit{failures}(P) \cap \{f\} = \mathit{failures}(P') \cap \{f\}$ then $\neg R(P')$, so we can set $\Phi = \{f\}$.

In our finite-alphabet case this topology coincides with the metric one since there only are finitely many failures and divergences with traces of length less than any chosen k. But the above (restricted to failures because of our initial assumption that we are only considering predicates of divergence-free processes) gives the following useful representation of closed predicates, noting that *a predicate is closed in this topology if and only if it is closed in the metric one.*

Choosing Φ for every P such that $\neg R(P)$, and setting $\Psi = \Phi \cap \mathit{failures}(P)$, it follows that if R is a closed predicate then there is a set Ref_R of pairs (Φ, Ψ) of finite sets of failures with $\Psi \subseteq \Phi$ such that

$$R(P) \equiv \forall (\Phi, \Psi) \in \mathit{Ref}_R . \mathit{failures}(P) \cap \Phi \neq \Psi$$

This observation leads to the following definition.

Definition 2.4 *If, for a predicate R, (Φ, Ψ) is such that*

$$\forall\, P . (\mathit{failures}(P) \cap \Phi = \Psi \Rightarrow \neg R(P))$$

then we say that (Φ, Ψ) is a refutation *for R. If Ref is a set of refutations which is sufficient to have a $(\Phi, \Psi) \in \mathit{Ref}$ refuting every P such that $\neg R(P)$, but no P with $R(P)$ is refuted by any member of Ref, then we will call it a a* complete set of refutations *for R. (A given predicate R may have many such sets representing it.)*

So Ref_R is a complete set of refutations for our R.

For example, in the case of the predicate $R(P) \equiv Q \sqsubseteq P$, a complete set of refutations is $\{(\{f\}, \{f\}) \mid f \notin \mathit{failures}(Q)\}$.

For the time being let us concentrate on a single refutation (Φ, Ψ) which is one member of a complete set Ref representing R. Then $\mathit{failures}(P) \cap \Phi = \Psi$ implies $\neg R(P)$. We can try to construct $F_{\Phi,\Psi}$ and $G_{\Phi,\Psi}$ such that

$$F_{\Phi,\Psi}(P) \sqsubseteq G_{\Phi,\Psi}(P) \qquad (\$)$$

if and only if $\mathit{failures}(P) \cap \Phi \neq \Psi$. One way of approaching this is to aim for the following

[4] The topologies defined in those papers are actually over possibly infinite product-spaces of processes. Since, for the time being at least, we are only interested in predicates of single processes, our definitions are a little simpler. In any case the significant differences in considering product spaces only appear in infinite ones, which are most unlikely to be of practical interest in refinement checking.

- Make $F_{\Phi,\Psi}(P)$ and $G_{\Phi,\Psi}(P)$ always equal one of $\Theta = e \to STOP$ and $\Xi = (e \to STOP) \sqcap STOP$ for e any fixed event outside the alphabets of the processes P we are considering (noting that Θ strictly refines Ξ).
- Make $F_{\Phi,\Psi}(P)$ equal Θ if and only if *failures*(P) contains no element of $\Phi \setminus \Psi$. (Equivalently, *failures*$(P) \cap \Phi \subseteq \Psi$.)
- Make $G_{\Phi,\Psi}(P)$ equal Ξ if and only if *failures*(P) contains every element of Ψ. (This is equivalent, thanks to our assumption that $\Psi \subseteq \Phi$, to *failures*$(P) \cap \Phi \supseteq \Psi$.)

Note that these specifications of $F_{\Phi,\Psi}$ and $G_{\Phi,\Psi}$ are both monotonic (and indeed both metric and partial-order continuous), meaning that they are reasonable things to aim to represent via CSP syntax. Importantly, refinement ($) only fails when $F_{\Phi,\Psi}(P) = \Theta$ and $G_{\Phi,\Psi}(P) = \Xi$, namely exactly when *failures*$(P) \cap \Phi = \Psi$.

Let us first consider how to build $F_{\Phi,\Psi}(P)$. What we would like is that it always has the initial capability to communicate the event e (and only e) but can only deadlock on the first step if *failures*(P) contains *any* element of $\Phi \setminus \Psi$. Let (s, X) be a typical element of this set. We define a "testing" process for it as follows:

$$\begin{array}{rcl} T(\langle\rangle, X) & = & ?x : X \to e \to STOP \\ T(\langle a\rangle\hat{}u, X) & = & a \to T(u, X) \\ & & \Box\ e \to STOP \\ & & \\ Test(s, X, P) & = & (T(s, X) \underset{\Sigma^-}{\|} P) \setminus \Sigma^- \sqcap (e \to STOP) \end{array}$$

where Σ^- is the set of all events other than e.

Since $T(s, X)$ only has finite traces, the hiding can introduce no divergence. It is obvious that the only possible traces are $\langle e\rangle$ and $\langle\rangle$, and that on the empty trace $Test(s, X, P)$ can only refuse e when its T is in state $(\langle\rangle, X)$. This is when it and P have each performed the trace s; but the combination then will only deadlock if P is refusing all the events in X. In other words the combination can deadlock on the empty trace precisely when P has the failure (s, X). If we now define

$$F_{\Phi,\Psi}(P) = \bigsqcap(\{e \to STOP\} \cup \{Test(s, X, P) \mid (s, X) \in \Phi \setminus \Psi\})$$

it follows that we have exactly the function we want, which is finite-state because Φ is finite.

The nondeterminism here means that we are running the various tests $T(s, X)$ as alternatives: if any one of them gives value Ξ then this disjunction does. The function $F_{\Phi,\Psi}(P)$ is distributive over $\sqcap$ since each particular run of it uses P at most once: this will be of interest later.

The situation with $G_{\Phi,\Psi}(P)$ is somewhat different, since in order for it to deadlock on $\langle\rangle$ we require P to have every failure in Ψ. We can fix this by replacing the nondeterministic choice above with interleaving:

$$\begin{array}{rcl} G_{\Phi,\Psi}(P) & = & ((STOP \sqcap e \to STOP) \mid\mid\mid (\mid\mid\mid_{(s,X)\in\Psi}\ Test(s, X, P))) \\ & & \underset{\{e\}}{\|}\ e \to STOP \end{array}$$

If P fails to have any one of the selected failures then, since the appropriate component of the interleaving above cannot refuse e on the empty trace, neither can the whole interleaving. If it does have all of Ψ then it can, but will certainly have the trace $\langle e\rangle$. The second line makes sure that nothing can happen after the first e, so that it always equals Θ or Ξ. So this definition does what we want; it is not, however distributive since $\mid \Psi \mid$ copies of P are run in parallel. Indeed, we shall see later on that there is no hope of $G_{\Phi,\Psi}$ being distributive.

Therefore we have the $F_{\Phi,\Psi}$ and $G_{\Phi,\Psi}$ we wanted to represent a single refutation. In order to check an arbitrary closed predicate, however, we may need to check an infinite number. The fact that we are considering only a finite alphabet, however, means that there are only (at worst) a countable infinity of them, and so we can list the whole of our set *Ref*:

$$(\Phi_0, \Psi_0), (\Phi_1, \Psi_1), \ldots$$

where the indices range over a set I which is either some $\{0 \ldots n\}$ or the set $\mathbb{N}$ of natural numbers. (Obviously there is no need to worry about the case where I is empty since then the predicate R is *true*.) We simply appeal to the ability demonstrated earlier to represent the conjunction of a countable family of refinements, and the entire closed predicate is represented.

Though this construction uses unbounded nondeterminism the resulting F and G do in fact satisfy (UC) because of the traces $\langle a, a, \ldots, a, b\rangle$ that are introduced: there are only finitely many refutations that are relevant to traces of any given length in the resulting $F(P)$ and $G(P)$. We can conclude the following.

Theorem 2.1 *The predicates of $\mathcal{N}^+$ which are closed in the metric topology are precisely those that can be expressed as a refinement $F(P) \sqsubseteq G(P)$ in which F is distributive, and where each of F and G satisfy (UC), but where the constructions of F and G are in general infinitary.*

Now consider the function $H(P) = a \rightarrow F(P) \sqcap b \rightarrow G(P)$, where F and G satisfy (UC) and represent the predicate R. Since the set of processes

$$C = \{a \rightarrow Q_1 \sqcap b \rightarrow Q_2 \mid Q_1 \sqsubseteq Q_2\}$$

is closed, and $H^{-1}(C)$ is precisely the set of processes satisfying R, we get the following result.

Theorem 2.2 *The closed sets in $\mathcal{N}^+$ are precisely the inverse images of C under uniformly continuous CSP-representable functions.*

(That all such inverse images are closed follows by a standard topological argument.)

This type of analysis does not give us much practical help in designing reasonable ways of expressing predicates for FDR, but it does suggest that most reasonable predicates can be expressed. More than anything else, the calculations above resemble those for a full abstraction result (ordinary ones for CSP being in [Ros98]), and it is indeed a result in the same spirit.[5] To be of practical use a predicate would need $F(P)$ and $G(P)$ to be finite-state if P is, as well as each having finite syntax; and the above certainly do not achieve this. This is inevitable given the generality of the above result, since the set of closed predicates is uncountable and the set of finitary (F, G) is countable. (This is because the number of finite pieces of CSP syntax is countable.)

In practical terms one will want to keep more careful control over the relationships between the traces being considered, both for different failures of the predicate and between the different copies of P that are running: we really do not want to start up several copies of P running to test every single potential failure of the predicate.

We now illustrate some techniques that can be used to do it more efficiently by means of an example. This is deliberately chosen so it has neither the *distributive* nor *refinement-closed* property which we will analyse later. That, however, makes it a little contrived since the author cannot think of straightforward predicates which have a naturally useful interpretation, and which have neither property.

Example 2.1 Consider the following predicate on a process P:

Each trace s of P which has any extension has exactly two: $s\hat{}\langle x\rangle$ and $s\hat{}\langle y\rangle$ with x and y being distinct elements of Σ which may vary with s.

The proof of Theorem 2.1 would suggest we have to look at many distinct finite sets of traces (modelled as the failures $(s, \emptyset)$) to decide this: all sets of the form $\{s\hat{}\langle x\rangle \mid x \in \Sigma\}$ whose intersection with P's traces must always have zero or two elements exactly. But in fact we can do a lot better than this. Imagine running three copies of P side by side, making sure that they maintain the same trace by some mechanism until we choose to test them. When we test them we must ensure that if there is any next event then the three copies between them have two but not three. On firing up a test (which will be done by a monitoring process M running in parallel with them) we can get M to communicate *twoevs* if it detects there are two options at this point (from two of the P's after the same trace) and *threeevs* if three. All we have to check then is that every trace of P that has a continuation can be followed by *twoevs* but never *threeevs*. The following uses the very useful trick of double renaming the copies of P so that as well as following events by their usual

[5] In fact the best way to regard our result is as a *higher order* form of full abstraction. The usual definition of (at least the main part of) full abstraction is that whenever two processes are distinguished by some model then there is a simple test on some context of the processes that distinguishes them. What we have here is a proof that, given a closed *set* of processes, we can used the same pair of contexts to distinguish between all pairs of processes inside and outside the set.

names they each have a separate tagged copy.

$$\begin{array}{rcl} CN(P) & = & ((rn(1,P) \underset{A}{\|} rn(2,P) \underset{A}{\|} rn(3,P)) \\ & & \underset{A \cup A.1 \cup A.2 \cup A.3}{\|} M) \setminus (A.1 \cup A.2 \cup A.3) \\ rn(i,P) & = & P[[a \leftarrow a, a \leftarrow a.i \mid a \in A]] \\ M & = & ?x : A \rightarrow M \\ & & \Box ?x.1 : A.1 \rightarrow ?y.2 : A.2 \rightarrow ?z.3 : A.3 \rightarrow \\ & & \quad ((card\{x,y,z\} = 2) \,\&\, twoevs \rightarrow STOP \\ & & \quad\; \Box\ (card\{x,y,z\} = 3 \,\&\, threeevs \rightarrow STOP)) \end{array}$$

Note here that all three copies of P keep in synchrony until the first one communicates in its alternative copy alphabet $A.1$ with M, which then tests to see what events the other two can do. The copy alphabets are hidden so we only see ordinary events plus the two signals.

To test the requirement we can cook up a slightly simpler process along the same lines.

$$\begin{array}{rcl} TT(P) & = & (rn(1,P) \underset{A \cup A.1}{\|} M')[\![a.1 \leftarrow twoevs \mid a \in A]\!] \\ M'(P) & = & ?x : A \rightarrow M'(P) \\ & & \Box ?a.1 : A.1 \rightarrow STOP \end{array}$$

So for each trace $s\hat{\ }\langle a \rangle$ of P we have the trace $s\hat{\ }\langle twoevs \rangle$ in $TT(P)$, but it never communicates *threeevs*.

What we actually now require is that $TT(P)$ and $CN(P)$ are trace equivalent: this is easily testable via two refinement checks, or indeed one if we form a binary conjunctions by prefixing the two sides with different events (here a and b) and test

$$\begin{array}{c} ((a \rightarrow TT(P)) \Box (b \rightarrow CN(P))) \| Chaos_\Sigma \\ \sqsubseteq \\ ((a \rightarrow CN(P)) \Box (b \rightarrow TT(P))) \| Chaos_\Sigma \end{array}$$

(The $Chaos_\Sigma$'s make sure that the refinement ignores refusals. Note that this binary conjunction uses a simpler recipe than the countable form described earlier.)

This particular predicate is neither refinement closed (because a refinement can reduce two continuations to one) nor distributive (since $\sqcap$ can increase the number of continuations).

We were able to make this check finitary because of the way that all the behaviours we were comparing at any one time always come from the same, arbitrary trace. These things are not essential – for example we could use any finitely computable property of traces to tell us which to look at – but they certainly help. The topic of exactly which predicates may be captured with F and G both finitary remains one for future research, though some progress is reported in [Ros04].

3. Distributive and refinement-closed predicates

We will now look at two special cases, namely refinement-closed and distributive predicates of $\mathcal{N}^+$, as defined in the introduction.

These are both natural attributes for useful CSP predicates to have: any instance in which a process P can satisfy anything like a functional specification, and some refinement P' does not, requires explanation, since P is permitted to behave exactly like P'. The proposer must answer the question of why it is reasonable for P to satisfy the specification if the totality of its actual behaviour does not. And in many cases we would expect that if P and Q both satisfy a specification then the process which behaves like either one of them also satisfies it. The following theorem demonstrates that these two properties together reduce a specification to a well-known case (a satisfiable predicate is one which is satisfied by at least one thing; it is not equivalent to *false*.).

THEOREM 3.1 *The satisfiable predicate R is refinement closed and distributive if and only if it is expressible as a simple refinement check.*

Proof. Suppose first that R has these properties. Let $\chi(R) = \bigsqcap R$ (where R is identified with the set of processes satisfying it), then $\chi(R) \in R$ by distributivity, and $\chi(R) \sqsubseteq P$ implies $P \in R$ by refinement-closure. It follows immediately that $P \in R$ if and only if $P \sqsupseteq \chi(R)$, as required.

Conversely, if $R(P)$ is specified by $P \sqsupseteq S$, the proofs that it is satisfiable (by S), refinement-closed and distributive are all trivial. $\square$

What we will do in this section is to consider predicates which are either refinement closed *or* distributive, and how they might be represented.

3.1. Predicates that are refinement-closed

Since each CSP context $F(P)$ and $G(P)$ is monotonic with respect to P, it follows immediately that whenever $F(P)$ is constant (not depending on P) then the check (†) is refinement closed. It is therefore reasonable to speculate, after the result in Theorem 2.1, that every closed and refinement-closed predicate can be expressed in this way. Since the notation $F(P)$ with F constant is a little confusing, we will in this subsection consider refinement checks of the form

$$R(P) \equiv \mathit{Spec} \sqsubseteq G(P) \qquad (\sharp)$$

It is well-known that all standard CSP operators other than recursion are *distributive*, in each of their arguments separately. (A context $H(\cdot)$ is distributive if $H(P \sqcap Q) = H(P) \sqcap H(Q)$ for all P and Q, and more generally $H(\bigsqcap S) = \bigsqcap\{H(P) \mid P \in S\}$ for all nonempty sets S of processes). If G is distributive it follows that the $R(P)$ of ($\sharp$) is itself distributive, since if $G(P) \sqsupseteq \mathit{Spec}$ for each $P \in S$, then

$$G(\bigsqcap S) = \bigsqcap\{G(P) \mid P \in S\} \sqsupseteq \mathit{Spec}$$

Note that in this case we can deduce, by Theorem 3.1, that the specification given us by ($\sharp$) can in fact be written in the form $\mathit{Spec}' \sqsubseteq P$: a simple refinement. It follows that in order to get any increase in power we need to look at non-distributive $G(P)$. A context will typically be non-distributive if a single behaviour of $G(P)$ depends on more than one behaviour of P, which will be because P is either run in parallel or in sequence with itself (possibly being operated on first). Running P in parallel with itself is more general, and in any case the sequence idea is not very useful where we cannot have termination, so we will concentrate on P in parallel with itself.

Recall the structures of the F and G used in the proof of Theorem 2.1: the distributive function $F(P)$ uses many instances of P, but they are all on distinct execution paths – no sequence of actions of $F(P)$ invokes any more than one of them in parallel. On the other hand $G(P)$ interleaves multiple copies of P, which leads to it not being distributive. To see why this happens consider $H(P) = P \mid\mid\mid P$ and observe that if $P = a \rightarrow P$ and $Q = b \rightarrow Q$ then

$$H(P \sqcap Q) \neq H(P) \sqcap H(Q)$$

since the left-hand side contains traces with both a's and b's, whereas the right-hand side does not.

What we now do is to prove what was hoped for at the start of this section:

Theorem 3.2 *Every closed and refinement-closed predicate can be expressed in the form* ($\sharp$).

Proof. If R is such a predicate and $\neg R(P)$ we know that there is k such that

$$\forall\, Q . Q \downarrow k = P \downarrow k \Rightarrow \neg R(Q)$$

Now suppose $R(Q)$ holds and $Q \downarrow k \sqsubseteq P \downarrow k$. The construction

$$\begin{aligned} \mathit{failures}(Q') \quad = \quad & \{(s, X) \in \mathit{failures}(Q) \mid \; \mid s \mid < k \wedge (s, X) \in \mathit{failures}(P) \\ & \qquad \vee \exists\, u, v.(s = u\hat{\ }v \wedge \; \mid u \mid = k \wedge u \in \mathit{traces}(P))\} \end{aligned}$$

then builds a process that refines Q and for which $Q' \downarrow k = P \downarrow k$. This is a contradiction since we can deduce $\neg R(Q')$ from the equality of restriction, and $R(Q')$ by refinement of Q. Therefore the same value of k actually yields

$$\forall\, Q . Q \downarrow k \sqsubseteq P \downarrow k \Rightarrow \neg R(Q)$$

It follows that there is a finite set of failures Φ, namely

$$failures(P) \cap (\{(s,X) \mid | s | < k\} \cup \{(s,\emptyset) \mid | s | = k\})$$

such that $failures(Q) \supseteq \Phi$ implies $\neg R(Q)$. (Φ is just the set of failures which are used in calculating $P \downarrow k$.)

But this is equivalent to having $\Psi = \Phi$ in the proof of Theorem 2.1, which will now yield the function $F_{\Phi,\Phi}$ for the left-hand side of the refinement check, if we are concentrating on this one set. But this is just the constant function mapping every P to $\Theta = e \rightarrow STOP$.

We can now build the complete representation Ref_R of R as a collection of such Φ's. There is of course no need to choose the Φ's that refute $R(P)$ using just restrictions $P \downarrow k$, provided that for every P with $\neg R(P)$ there is one with

$$\Phi \subseteq failures(P) \wedge \forall\, Q.\Phi \subseteq failures(Q) \Rightarrow \neg R(Q)$$

(and no other Φ's than ones with this property). Let S be a set of Φ's that characterise R in this way.

Since we have the constant Θ on the left-hand-side of our refinement check for every $\Phi \in R$, all we need to do is to check that none of the $G_{\Phi,\Phi}(P)$'s can deadlock on the first step. We can therefore dispense with the form of conjunction which uses separating traces and define, for ($\sharp$),

$$\begin{array}{rcl} Spec & = & e \rightarrow STOP \\ G(P) & = & \bigsqcap_{\Phi \in S} G_{\Phi,\Phi}(P) \end{array}$$

which will generate a predicate equivalent to R.

Unlike the case where the different pairs (Φ, Ψ) were kept apart by different-length traces, this $G(P)$ may not satisfy (UC) since it makes use of nondeterministic choice in a way that can potentially map arbitrarily long behaviours onto the initial deadlock. Of course if (UC) were required one could revert to the use of a's and b's. $\square$

If we can bound the size of the sets Φ, giving a measure of how many different behaviours we need to observe at once to refute our predicate R, then a simpler and more practical form of predicate can be found. A set of pairs $\{(\Phi_i, \Psi_i) \mid i \in I\}$ which characterises a predicate in the sense of Theorem 2.1 will be said to be k-bounded if each Ψ is no larger than k. This is rather more striking for the representations used in Theorem 3.2 since each $\neg R(P)$ can be detected by observing that one of a selection of bounded sets is a subset of P.

An excellent example of this is the determinism condition described in the introduction: this fails just when P has the failures

$$(s, \{a\}) \quad \text{and} \quad (s\hat{}\langle a\rangle, \emptyset)$$

for some s and a. So this predicate is 2-bounded and every failure of it shows up in P having an offending pair of behaviours. The way to test for this is to run two copies of P together, with separated alphabets, and ensure that one of them cannot refuse an event that the other accepts on the same trace. This suggests the model of running the two copies interleaved, probed and tested by a monitor process that observes their behaviours and flags errors.

The best way of testing in this way for determinism is for the monitor M to accept any event from the first copy of P and then force the second to accept it. If this is successful it does the same again, otherwise P was nondeterministic.

To implement this we run $P[\![a \leftarrow a.1 \mid a \in A]\!]$, and $P[\![a \leftarrow a.2 \mid a \in A]\!]$ (similarly to the example), where A contains all events of P, interleaved, and put them in parallel with

$$Monitor = ?x.1 : A.1 \rightarrow x.2 \rightarrow Monitor$$

and the monitor interface $MI = A.1 \cup A.2$. In this the $Spec$ to refine $G(P)$ against is just the specification that $G(P)$ never deadlocks on an odd-length trace (with this formulation also taking into account that $A.1$ and $A.2$ events always alternate):

$$\begin{array}{rcl} Spec & = & STOP \sqcap \bigsqcap_{a \in A} a.1 \rightarrow Spec' \\ Spec' & = & \bigsqcap_{a \in A} a.2 \rightarrow Spec \end{array}$$

This implementation of determinism testing (different from the one used by FDR which is non-monotonic and outside the scope of this paper) was invented by Lazić [Laz98]. That representation was the starting point for the work in this paper.

One can modify this test for determinism to avoid the renaming[6]: choose an event *clunk* not used by P and define

$$\begin{aligned}
Clunks &= ?x : \Sigma \setminus \{clunk\} \rightarrow clunk \rightarrow Clunks \\
Clunk(P) &= P \underset{\Sigma\setminus\{clunk\}}{\parallel} Clunks \\
Repeat &= ?x : \Sigma \setminus \{clunk\} \rightarrow x \rightarrow Repeat \\
DetTest &= ((Clunk(P) \underset{\{clunk\}}{\parallel} Clunk(P)) \setminus \{clunk\}) \underset{\Sigma}{\parallel} Repeat
\end{aligned}$$

and this will also deadlock on an odd-length trace just when P has a nondeterminism.

Since we can make processes like M and *Repeat* monitor the behaviours of various copies of P on different traces as well as when they are constrained to act the same, it follows that the technique can simultaneously look for all misbehaviour in any k-bounded representation of a refinement-closed predicate using k interleaved renamed copies of P. Of course, unless the resulting M is itself finite-state, the implementation of the predicate will not be either, but the fact that P is used only k times means that M is the only possible source of infinitary behaviour. In the case where M is finite-state (with $\#M$ states, say), then the overall complexity of the refinement check is bounded by $\#M.N^k$ where P has N states.

3.2. Distributive predicates

Any predicate that can be written

$$R(P) \equiv F(P) \sqsubseteq G(P)$$

with G distributive is itself distributive since, if $R(P)$ and $R(Q)$ then

$$F(P \sqcap Q) \sqsubseteq F(P) \sqcap F(Q)$$

by monotonicity of F,

$$F(P) \sqcap F(Q) \sqsubseteq G(P) \sqcap G(P)$$

by $R(P) \wedge R(Q)$ and properties of partial orders, and $G(P) \sqcap G(Q) = G(P \sqcap Q)$ by distributivity of G. It follows that $R(P \sqcap Q)$ holds.

We might hope, after our experience to date, that any distributive predicate can be expressed in the above form. It is interesting here that we do not require F to be distributive for R to be.

Remember our earlier construction to represent the disjunction of two predicates. Since the disjunction of two distributive predicates is not in general distributive, the connection shown above between distributive G and distributive predicates shows that the non-distributive G remarked on earlier (involving interleaving) is in fact inevitable. (For example, if Q_1 and Q_2 are incomparable processes under refinement, the predicate $Q_1 \sqsubseteq P \vee Q_2 \sqsubseteq P$ is not distributive even though the two disjuncts obviously are.)

Suppose we are trying to represent a closed, distributive predicate R. Since our predicate is closed there must be a complete set Ref_R of refutations (Φ, Ψ) which characterise it, from the definition of the δ-topology. We can also assume that all are *minimal*, in the sense that none is trivially implied by another refutation of R (whether inside or outside Ref_R) in the following sense.

We can write $(\Phi, \Psi) \Leftarrow (\Phi', \Psi')$ if $\Phi \subseteq \Phi'$ and $\Phi \cap \Psi' = \Psi$. In that case, if $failures(P) \cap \Phi' = \Psi'$ then certainly $failures(P) \cap \Phi = \Psi$. So if (Φ, Ψ) is a refutation for R there is no point in including (Φ', Ψ'): it is non-minimal and it would be better to include only (Φ, Ψ).

Note that this is not quite the same as assuming that the set of refutations Ref_R is itself subset minimal.

Claim that any minimal (Φ, Ψ) has $\mid \Psi \mid \leq 1$. If not we can find one (Φ, Ψ) such that $\mid \Psi \mid > 1$. Minimality of (Φ, Ψ) tells us that, for each $\Gamma \subset \Phi$, there is some Q_Γ such that (i) $R(Q_\Gamma)$ and (ii) $failures(Q_\Gamma) \cap \Gamma = \Psi \cap \Gamma$.

[6] The renaming is awkward in FDR because many extra channel declarations are required in general.

(If this were not the case then $(\Gamma, \Psi \cap \Gamma)$ would be a refutation that implies (Φ, Ψ).) Let $\mathcal{G}$ be the set of all Γ that have $\Phi \setminus \Psi \subseteq \Gamma$ and $\mid \Gamma \cap \Psi \mid = 1$. (Namely, $\mathcal{G} = \{(\Phi \setminus \Psi) \cup \{f\} \mid f \in \Psi\}$.)

Consider $Q^* = \bigsqcap\{Q_\Gamma \mid \Gamma \in \mathcal{G}\}$. (Note that all such Γ are proper subsets of Φ by our assumption that $\mid \Psi \mid > 1$.) Since R is distributive we have $R(Q^*)$, and by construction

- $\Phi \setminus \Psi \subseteq \mathit{failures}(Q^*)$ – as this is true for each individual Q_Γ, and
- $\Psi \cap \mathit{failures}(Q^*) = \Psi$ – since for each $(s, X) \in \Psi$ there is $\Gamma \in \mathcal{G}$ with $(s, X) \in \Gamma$.

It follows – by the fact that (Φ, Ψ) are assumed to refute R – that we have a contradiction to the assumption that $\mid \Psi \mid > 1$.

We therefore need only consider the cases of (Φ, Ψ) where Ψ is empty or a singleton set. If Ψ is empty it means that for $\mathit{failures}(P)$ to contain no member of the corresponding Φ implies $\neg R(P)$. On the other hand if $\Psi = \{(s, X)\}$ then it means that if $R(P)$ holds and $(s, X) \in \mathit{failures}(P)$ then $\mathit{failures}(P)$ must contain at least one member of $\Phi \setminus \Psi$.

We can actually guarantee that each Φ occurs in our Ref_R, with its minimised elements, only with $\Psi = \emptyset$ or only with singletons. For if $(\Phi, \emptyset), (\Phi, \{f\}) \in \mathit{Ref}_R$ then clearly $(\Phi \setminus \{f\}, \emptyset)$ refutes R and is less than both. (Whether f were there or not, R would be refuted by one or the other.)

It is easy to see that any R represented by such a Ref_R is distributive: for if $P \sqcap Q$ were refuted by some $(\Phi, \emptyset)$ then it is clear both P and Q are – neither can contain any member of Φ. And if it were refuted by $(\Phi, \{f\})$ then (as nondeterministic choice is union) one of P and Q must contain the failure f, and neither can contain any member of $\Phi \setminus \{f\}$, meaning that the one containing f fails R. In either case we have $\neg R(P \sqcap Q) \Rightarrow \neg R(P) \vee \neg R(Q)$ which is logically equivalent to

$$R(P) \wedge R(Q) \Rightarrow R(P \sqcap Q)$$

namely distributivity. (A slightly altered form of this argument also works for infinite nondeterministic choice.)

If Ref_R contains only members of the form $(\Phi, \emptyset)$ then R is closed under anti-refinement – not such an intuitive concept as being closed under refinement! On the other hand if it contains at least one of the form $(\Phi, \{f\})$ then it is not (by minimality of the members of Ref_R and a simple argument). Notice also that unless Ref_R contains at least one $(\Phi, \{f\})$ with $\Phi = \{f\}$ then Chaos_Σ (the refinement-least element of $\mathcal{N}^+$) will satisfy R. A simple refinement specification (which is distributive as observed earlier) is precisely one that can be represented by Ref_R containing *only* pairs of the form $(\{f\}, \{f\})$.

Just as the concept of k-boundedness was so important in the representation of refinement-closed predicates as more convenient refinement checks, it should not be surprising that the shape of Ref_R plays a large role in the representation of distributive ones.

It is cleaner to separate out Ref_R into two sets: C_1 being all the pairs with empty Ψ and C_2 being those with singleton Ψ. We can then verify R by two refinement checks, one for each.

So consider first the case of a collection C $(= C_1$, say) of pairs with empty Ψ. It is a corollary to the proof of Theorem 2.1 that the following theorem below holds. (Each $G_{(\Phi,\emptyset)}$ is the constant Ξ meaning that if Ref_R is infinite then the G produced by the proof of Theorem 2.1 is the process S which performs any number of a's nondeterministically, one b, and then equals Ξ. If C is finite then the same S will still work with $F(P)$ being the process formed by a selection of the $a^n b$ traces being followed by one of the $F_{(\Phi,\emptyset)}(P)$'s as $(\Phi, \emptyset)$ varies over C; all the rest being followed by Ξ. Alternatively (when C is finite) one can choose a simpler and finite S.)

THEOREM 3.3 *The closed R is anti-refinement closed, or equivalently can be refuted using a set of pairs (Φ, Ψ) such that all Ψ's are empty, if and only if*

$$R \equiv F(P) \sqsubseteq S \qquad (\flat)$$

for some CSP-definable F satisfying (UC), and fixed process S. Furthermore, in the "only if" case we can choose F to be distributive.

For $C = C_2$ consisting of pairs with singleton Ψ we observe that a process satisfies the equivalent predicate (i.e., the one for which C is a complete set of refutations) if and only if

$$\mathit{failures}(P) = \mathit{failures}(P) \setminus \{f \mid (X, \{f\}) \in C \wedge X \cap \mathit{failures}(P) = \emptyset\}$$

and that this really only means testing left-to-right inclusion. If we could express the right-hand-side as a process, then this would be a refinement check. But the right-hand side is not necessarily a process, since it may well not satisfy the model healthiness conditions[7] (meaning that there is no hope of expressing it in CSP). We can, however, do an equivalent trick: let e be an event not appearing in P or Φ, and consider $P' = P \underset{\Sigma\setminus\{e\}}{\|} OneE$ where

$$OneE = (?x : \Sigma \setminus \{e\} \to OneE) \mathrel{\Box} (STOP \sqcap e \to STOP)$$

$P' \sqsubseteq P$, so in particular P' has all of P's failures. The big difference, however, is that we can remove any failure $(s, X \cup \{e\})$ without harming the model axioms, and so we can let

$$\overline{P{*}C} = \left(\begin{array}{lll} \mathit{failures}(P') & \setminus & \{(s, X \cup \{e\}) \mid (\Phi, \{(s, X)\}) \in C \\ & & \wedge (\Phi \setminus \{(s, X)\}) \cap \mathit{failures}(P) = \emptyset\} \end{array} \right)$$

and this always is a process. It removes from P' just those $(s, X \cup \{e\})$ for which the complete set C of refutations says that (s, X) may not be in P. This means we can check our predicate via the refinement

$$\overline{P{*}C} \sqsubseteq P' \qquad (\natural^*)$$

Of course this is just one possible trick for expressing the predicate of C as a refinement.[8] And to use it we still have to implement $\overline{P{*}C}$.

So how can we build $\overline{P{*}C}$? Obviously it must have all traces of P, with the possible addition of e (and nothing further) after each of them. We must give it all the failures of P (aside from refusal of sets including e), and we must allow the presence of sets involving e to be influenced by C and its observations of $\mathit{failures}(P)$.

One way of achieving this is by the parallel composition

$$((P \underset{\Sigma\setminus\{e\}}{\|} Q) \mathrel{|||} \mathit{Chaos}_{\{e\}}) \underset{\Sigma}{\|} OneE$$

where Q is a process whose traces include those of P and whose failures on each trace are a subset of those of P (if the trace is one of P). If Q does not have the failure $(s, X \cup \{e\})$, which is achieved by making Q always offer e when it refuses X after s, then the above combination does not have the failure $(s, X \cup \{e\})$, but since P' does if (s, X) is a failure of P, the refinement $(\natural^*)$ would then fail.

So we would want Q to have the failure $(s, X \cup \{e\})$ only if all of the $(\Phi, \{(s, X)\}) \in C$ (with this failure) have at least one member of $\Phi \setminus \{(s, X)\}$ in $\mathit{failures}(P)$. This can be achieved (less clumsily in the case where the number of $(\Phi, \{f\})$ with a given f is bounded), but not in ways which seem to the author to lead to likely practical implementations.

It is appropriate to widen the specific check $(\natural^*)$ above to the more general one below, which is closer to our previously named styles.

Theorem 3.4 *Every distributive closed predicate on $\mathcal{N}^+$ can be represented by a check of the form*

$$F(P) \sqsubseteq P' \qquad (\natural)$$

where P' is as above and F is a CSP context.

In practice it is often possible to simplify this form with P' becoming P. It is an open question whether it can *always* be simplified like this.

It is, of course, interesting that distributive predicates can be expressed in this form. Notice that we have now proved the following:

- Starting from the observation that every behavioural predicate (i.e., one which is refinement-closed and distributive) can be expressed as a simple refinement $\mathit{Spec} \sqsubseteq P$,

[7] Such conditions, for example the prefix-closure of the traces of a process, are common to all CSP models. See [Ros98], for example. The problem we have here is that there is no good reason why the right-hand-side of the above equation is, in general, a process which satisfies these conditions.

[8] To be even more devious, one could use the same trick to code $(\Phi, \emptyset)$ pairs by converting them into $(\Phi \cup \{(\langle\rangle, \emptyset)\}, \{(\langle\rangle, \emptyset)\})$ on the grounds that *every* process has the failure $(\langle\rangle, \emptyset)$. It is up to an individual as to whether to use this or not, but we will implicitly do so when claiming that *every* distributive predicate can be expressed in forms $(\natural^*)$ and $(\natural)$.

- replacing P by a general context allows us to check any closed and refinement-closed predicate,
- whereas replacing *Spec* by a general context, and sometimes P by P', allows us to check any closed and distributive predicate, and finally
- making both the left- and right-hand sides arbitrary CSP contexts allows us to check any closed predicate.

4. Examples of distributive and refinement-closed predicates

This paper was inspired by the author's contemplation of a number of examples, each of which was either distributive or refinement closed. The crucial thing was that in each case it was possible to find a representation as refinement even where this seemed (at the time) unlikely. We have already seen determinism above, whose refinement representation was discovered by Ranko Lazić.

A closely related predicate is that of noninterference, as specified in [Ros95, Ros98]. The impossibility of things which a high level process with alphabet H transmitting information through a system P to a low-level one with alphabet L is expressed as

$$\mathcal{L}_H(P) \quad \text{is deterministic}$$

where $\mathcal{L}_H(P)$ is the *lazy abstraction* operator which provides the view in $\Sigma \setminus H (= L)$ on the assumption that there is an arbitrary but unknown process interacting in an unseen way with P in the alphabet H.

Given that $\mathcal{L}_H(P)$ is defined in [Ros98] to be the divergence-free process whose failures are the stable ones of

$$(P \underset{H}{\parallel} Chaos_H) \setminus H$$

(namely, the failures generated at stable, or τ-free, states as opposed to ones introduced by divergence-strictness) it becomes straightforward to use Lazić's technique to convert noninterference (a closed and refinement-closed predicate) into a refinement check: just substitute the above representation of abstraction into the determinism check given earlier, and use the failures refinement check of FDR. The latter is done because it ignores any divergences which this syntactic representation of abstraction introduces.

In fact this representation of the noninterference check is arguably better than using the FDR determinism check implemented at the time of writing, because it (unlike the FDR one) is never corrupted by one of these divergences. (Using the FDR determinism check on the process above will sometimes fail to find a result because of one of these, though there are ways around this by using different representations of the predicate.)

A second predicate is that of *fault tolerance*, as formulated in [Ros98] for systems in which errors are triggered by some events that are assumed to be in the hands of some "demon" rather than ordinary users. One advantage of this approach is that it allows assumptions to be made on the number, frequency, etc. of errors thanks to parallel composition of the system P with some error regulator, creating an overall system P_{reg}. A second is that it gives a very elegant characterisation of fault tolerance:

$$P \underset{E}{\parallel} STOP \sqsubseteq \mathcal{L}_E(P_{reg})$$

where E is the set of fault events. This simply says that the system with faults permitted but abstracted is a refinement of how the system behaves when there are no faults.

Intuitively this is similar to non-interference, since it says that error events do not significantly affect what the user sees. It is, however, a different predicate since it allows errors to cut down the nondeterministic range of behaviour, and indeed allows nondeterminism much more freely in general. Unlike noninterference it is not refinement-closed, but as the form of the above definition shows (given our earlier analysis) it is distributive (which noninterference is not). The right-hand side will always be a distributive context, as the regulator process if any should not involve P.

A little transformation easily converts the above into the form ($\natural$) in the case where $P_{reg} = P$:

$$(P \underset{E}{\parallel} STOP) \mid\mid\mid Chaos_E \sqsubseteq P$$

More recently the author was presented with two predicates representing correct interaction between client/server pairs, developed by Joy Reed and Jane Sinclair. Here it turned out that one was distributive but not refinement closed, and the other (which is its weakest refinement-closed strengthening) is not distributive.

His experience in representing these as refinements was similar to that for the above examples, as is reported in [RSR04]. Some further examples of natural non-behavioural specifications can be found in [Ros04].

5. Divergence and related matters

We showed earlier (Theorem 2.1) that any predicate represented using (†) for F and G satisfying (UC) is topologically closed. We also observed that hiding, either coupled with infinite nondeterministic choice or used in a way that might introduce divergence, had the potential to create functions not satisfying (UC).

They can certainly create predicates that are not closed: consider

$$STOP \quad \sqsubseteq \quad P \setminus \{a, b\}$$

$$\bigsqcap\{(P \parallel nupto(n, b)) \setminus (\Sigma \setminus \{b\}) \mid n \in \mathbb{N}\} \quad \sqsubseteq \quad b \to STOP$$

where $nupto(n, b)$ is a process that does n events and stops, and additionally stops after communicating b. The first of these refinements specifies, thanks to König's lemma and Σ being finite, that P has only finitely many traces, all of which consist only of a and b. The second says that P can communicate a b at some point. Both of these predicates fail to be closed: for example each is satisfied by all the processes which perform n a's and then a b, but not by the limit of this sequence (the process that performs a's for ever).

Once we try to extend beyond the world of closed predicates it seems to be the case that the expressibility of predicates on (finite) traces and refusals starts to differ. For example, we saw above that the predicate "can communicate b" (a trace property) can be expressed as a refinement. However a similar failures predicate "there is a trace s after which b cannot be refused" is not expressible. For suppose it were expressed by $F(P) \sqsubseteq G(P)$. Consider the processes B^* which can refuse b after any length of trace of b's, and B_k which can do the same except not refusing b after k b's. Plainly we must have $F(B_k) \sqsubseteq G(B_k)$ but not $F(B^*) \sqsubseteq G(B^*)$.

The failure of refinement must be in refusal sets, since the traces and divergences of $F(B_k)$ must be the same as those of $F(B^*)$ and similarly for G. It follows that $G(B^*)$ must have some failure (s, X) which is not in $F(B^*)$ and hence (a) not in any $F(B_k)$ (as these all refine $F(B^*)$) and thus (b) not in $G(B_k)$ for any k either. We can be sure that s is a trace of both sides but not a divergence. It is a property[9] of CSP operators that, whenever (s, X) is a non-divergent failure of $H(P)$, then there is a finite set Φ of failures of P such that $(s, X) \in failures(H(Q))$ whenever $\Phi \subseteq failures(Q)$. It follows that "$(s, X) \in failures(B^*)$" is proved by some finite set of B^*'s failures, which means that it belongs to $G(B_k)$ for sufficiently large k. This contradicts what we already know. It follows that this predicate is *not* expressible via (†). (Note that this is the first such example we have seen.)

The author has identified two reasons for the comparative difficulty in expressing refusal predicates. The first, seen above, is that since refusals are only ever used to calculate other refusals, we cannot make use of (for example) divergences to give us extra leverage. The second is that we can only look at one refusal set per trace, and this is of uniformly bounded size thanks to the finite size of Σ. This is in contrast to traces, where we can string an infinite collection together and then hide them.

As an example illustrating the power of combining an infinite set of traces into one, consider the predicate "Q has a trace which is not a trace of P" for any fixed Q. We can assume Q has an infinite (necessarily countable) set of traces since otherwise it is a closed predicate and therefore representable. Since Q has a countable set of traces we can enumerate them $t_1, t_2 \ldots$ and build a process TRS_Q which performs these traces, in this order, putting an extra event *reset* after each. (For example, if Q can do any number of a's, then TRS_Q might perform no a's, *reset*, one a, *reset*, two a's, *reset*, three a's, and so on.)

Now consider the construct (taken from [Hoa85]):

$$Resettable(P) = P \bigtriangleup reset \to Resettable(P)$$

which can perform any sequence of P's traces separated by *reset*'s, and

$$AllTr_Q(P) = (TRS_Q \underset{\Sigma}{\parallel} Resettable(P)) \setminus \Sigma$$

[9] This follows from the continuity of all operators over the subset order on the stable failures model $\mathcal{F}$ of CSP: see chapter 8 or [Ros98]

$AllTr_Q(P)$ can diverge if and only if P has all of Q's traces, or in other words if P fails the predicate. Therefore testing if this context of P refines $STOP$ (the only value other than **div** for a process with all events hidden) decides our predicate, which is the negation of the closed

$$P \sqsubseteq_T Q$$

This example shows clearly how we can put infinitely many behaviours of P into a single divergence of $AllTr_Q(P)$. There is no analogue of this construction for failures or divergences.

For these reasons we will consider only trace predicates in the rest of this section, namely ones which are determined by the set *traces*(P) of P's finite traces. We saw earlier in this paper how to express binary disjunction of two failures predicates by means of doubling the events and using interleaving ($|||$). There is no way in which this approach can work for infinite disjunctions, because:

- it would require an infinite alphabet, which we have excluded, and more importantly
- infinite parallel operators are not properly definable in CSP since they cause great theoretical problems.

Let us consider the disjunction of refinement-closed closed trace predicates. The construction used in the proof of Theorem 2.1 to demonstrate that closed predicates can be represented as refinements uses failures refinement essentially, even in the case where what is being tested is a trace predicate. This was because we could use either traces or refusals to trigger the refusal of the event e and so giving Ξ rather than Θ, but this would not have been possible in the refinement case if Ξ and Θ had differed in traces. *If* we had been interested only in traces predicates earlier, then it would have been straightforward to have used $\Theta = STOP$ and $\Xi = e \to STOP$ in modified constructions to get the same representation results as Theorems 2.1, 3.2, 3.3 and 3.4. In particular any refinement-closed closed predicate would be representable by a check of the form

$$STOP \sqsubseteq_T G(P)$$

in which $G(P)$ can only take the two values $STOP$ and $e \to STOP$.[10]

If we have a countable collection of these predicates then it is possible to represent each of them like this, or to modify the construction so that when the kth of them fails the resulting $F_k(P)$ has the traces of $nas(k, e)$; $STOP$ (in other words it communicates k a's before a single e. (When it succeeds it only has the empty trace.) Now consider the process

$$H^*(P) = \bigsqcap\{F_k(P) \mid k \in \mathbb{N}\}$$

(using the latter form of F_k) which then has the traces of $A^*E = \bigsqcap\{nas(k, e) \mid k \in \mathbb{N}\}$ if and only if all the constituent refinements fail. Hence

$$STOP \sqsubseteq AllTr_{A^*E}(H^*(P))$$

is equivalent to the disjunction of the individual refinements.

Topologically speaking, this actually implies that *all* refinement-closed open sets of the traces model (i.e., the complements of closed sets) are expressible. For every refinement-closed open set U is the union of the closed and open (clopen) sets[11]

$$\{P \mid P \cap \Sigma^n = F\}$$

for those finite F and n such that the above set is contained in U.[12] Since there are only countably many of these "basic clopen sets" it follows that every open set is the union of countably many closed ones.

For the rather less useful category of anti-refinement-closed trace predicates we can get a similar result.

[10] It is of course natural that we should be able to judge trace predicates by means of trace refinement. However bear in mind below that we will make critical use of divergence to help judge trace predicates, and that in these cases we need trace-plus-divergence refinement, which can easily be judged in the failures-divergences model. If the presence or absence of refusal sets were an issue (which it will typically not be below since we hide all events) then parallel composition with $Chaos_\Sigma$ on the left-hand side of the refinement gives the proper meaning.

[11] These sets are, topologically speaking, a *basis*.

[12] If Q is in U then the processes $Q \uparrow n = Q \cup \{s\hat{\ }t \mid s \in Q \wedge \mid s \mid = n \wedge t \in \Sigma^*\}$ are a sequence that converges to it; therefore one of them is in U and all that $Q \uparrow n$'s refinements are too.

We know from earlier discussion that any anti-refinement-closed closed predicate can be represented in the form

$$F(P) \sqsubseteq S$$

However it is clear that if we are simply restricting ourselves to traces predicates then the refinement above can be trace refinement. This gives us an opportunity to transform it into the form

$$AllTr_S(F(P)) \sqsubseteq \mathbf{div}$$

(judged over the failures-divergences model $\mathcal{N}$) since the left-hand side diverges just when P has all S's traces. Now any nonempty set of such refinements

$$\{F_\lambda(P) \sqsubseteq S_\lambda \mid \lambda \in \Lambda\}$$

can be disjoined

$$\bigsqcap\{AllTr_{S_\lambda}(F_\lambda(P)) \sqsubseteq \mathbf{div}$$

This is the disjunction since the refinement plainly holds precisely when the left-hand side can diverge on $\langle\rangle$, which is equivalent to at least one of the constituent refinements being true.

This trivially implies that *all* anti-refinement closed trace predicates are expressible whether open, closed or neither. We can summarise the above results (plus re-statements and easy re-workings of some from earlier in the paper) as follows.

Theorem 5.1 *The following hold for traces predicates.*

- *All closed predicates can be decided by* (†) *using traces refinement* $\sqsubseteq_T$ *using functions satisfying* (UC). *We can similarly replace* $\sqsubseteq$ *by* $\sqsubseteq_T$ *for the other results of Sections 2 and 3, namely Theorems 2.1,2.2,3.1 3.2, 3.3 and 3.4.*
- *The classes of traces predicates representable are closed under countable conjunction and finite disjunction.*
- *Any countable disjunction of refinement-closed and closed traces predicates is representable via a refinement over the failures/divergences (or alternatively the little-used traces/divergences) model.*
- *Any anti-refinement-closed traces predicate is representable via a failures/divergences (or traces/divergences) refinement.*

It would be useful to extend the result about closure under countable disjunction to more than the closed refinement-closed predicates. However the author has been unable to find a construction which achieves this, under the conventions established in this paper of using any (potentially infinite) CSP construct over the failures/divergences model. However it is possible if we allow ourselves to move to the failures/divergences/infinite traces model $\mathcal{U}$ (see [Ros93, Ros98]). This is arguably a more natural model given the types of infinitary constructs we tend to use, but does of course take us further from the realms of practicality.

Consider the following alternative formulation of binary disjunction: suppose we have a pair of refinements $F_i(P) \sqsubseteq_T G_i(P)$ $(i \in \{1,2\})$. Then the process $G_\vee(P) = G_1(P) \triangle\ reset \rightarrow G_2(P)$ (for *reset* a new event as before) can perform any trace of $G_1(P)$, a *reset*, and then any trace of $G_2(P)$. We can use essentially the same trick as we did earlier with $|||$ and define

$$\begin{array}{rcl} F_\vee(P) & = & (Chaos_{\Sigma\setminus\{reset\}} \triangle\ reset \rightarrow F_2(P)) \\ & & \sqcap\ (F_1(P) \triangle\ reset \rightarrow Chaos_{\Sigma\setminus\{reset\}}) \end{array}$$

Essentially the same argument used with the interleaving version shows that

$$F_\vee(P) \sqsubseteq_T G_\vee(P)$$

is equivalent to the disjunction of the two constituents. This does not work for failures since the refusal at the end of the first process's trace is ignored.

It is possible to extend this to countable disjunction, but only if we use a model which admits infinite traces. For a sequence $\mathbf{P} = \langle P_i \mid i \in \mathbb{N}\rangle$ of processes we define

$$\Pi(\mathbf{P}) = head(\mathbf{P}) \triangle\ reset \rightarrow \Pi(tail(\mathbf{P}))$$

to be the process which performs one trace of each of the P_i in sequence, separated by *reset*'s. Suppose we are given sequences of functions $\langle F_i(\cdot) \mid i \in \mathbb{N}\rangle$ and $\langle G_i(\cdot) \mid i \in \mathbb{N}\rangle$ whose refinements over $\mathcal{U}$ represent a sequence of finite-trace predicates where any failure of refinement would show up as an illegal finite trace of the right-hand side. We can then define $\mathbf{S}_i$ to be the sequence of processes which is $Chaos_{\Sigma\setminus\{reset\}}$ except at position i, where it is $F_i(P)$ (P being the process we are checking for the disjunction of the implied predicates), and $\mathbf{I} = \langle G_i(P) \mid i \in \mathbb{N}\rangle$. Then

$$Chaos_\Sigma \underset{\Sigma}{\parallel} \bigsqcap\{\Pi(\mathbf{S}_i) \mid i \in \mathbb{N}\} \sqsubseteq_{\mathcal{U}} \Pi(\mathbf{I})$$

if and only if any one of the constituent refinements $F_i(P) \sqsubseteq_T G_i(P)$ holds. The "if" part is trivial, for if $Chaos_\Sigma \underset{\Sigma}{\parallel} F_i(P) \sqsubseteq_{\mathcal{U}} G_i(P)$ then

$$Chaos_\Sigma \underset{\Sigma}{\parallel} \Pi(\mathbf{S})_i \sqsubseteq_{\mathcal{U}} \Pi(\mathbf{I})$$

For "only if", observe that if none of $F_i(P) \sqsubseteq_T G_i(P)$ holds then we can find finite traces s_i of $G_i(P)$ which are not possible for $F_i(P)$. The trace $s_0\hat{}\langle reset\rangle\hat{}s_1\hat{}\langle reset\rangle \ldots$ is then a trace of $\Pi(\mathbf{I})$ which is not a trace of any of the constituents of the left-hand side of the refinement above.

The fact that we need to assume that the constituent refinements are themselves judged over $\mathcal{U}$ is not a damaging restriction since all of the constructions used earlier would have worked equally well over the more elaborate model.

Observing that any open predicate over the traces model is the countable disjunction of clopen ones (since there are only countably many clopen balls of the form $B(r, y) = \{x \mid d(x, y) < r\}$ in total, d being the metric with the same definition as earlier, but based on the standard restrictions $P \downarrow n$ over the trace model $\mathcal{T}$ rather than $\mathcal{N}$) and therefore representable using the above and Theorem 5.1, we get the following result.

THEOREM 5.2 *The following are representable as refinements over $\mathcal{U}$:*

(a) *Any countable disjunction of trace predicates representable as refinements over $\mathcal{T}$.*

(b) *All open traces predicates.*

Unlike the construction involving $AllTr_Q(P)$ above, it does not appear to be possible to map this infinite-trace refinement down to the failures/divergences model by using hiding to create divergences. The reason for this is that the left- and right-hand sides of our refinement are too complex, and in particular there are potentially uncountably many different counter-example traces which might be important. (If there were only countably many we could use similar constructions to the earlier case and the countable construction given earlier.)

Since $\mathcal{U}$ really is a much richer model than the ones based on finite traces it should perhaps not come as a great surprise that we seem to be able to express richer predicates in it. An interesting question that arises is whether one can express over it the countable disjunction of predicates based on both finite and infinite traces (there being no hope for refusals because the arguments above still apply). Certainly the above idea does not work, even for pairwise disjunction, as the construct never combines two infinite traces into a single one. The interleaving-based pairwise disjunction from earlier in this paper does work, however.

The author believes it is possible to give a countable disjunction using a combination of the ideas in these two methods, but the result is even more esoteric than the ideas we have seen before. Simple infinite interleaving will not work because (i) this construct is itself inadmissible, like all infinite parallel operators in CSP and (ii) the actions of individual processes cannot be distinguished without using an infinite alphabet.

If, on the other hand, we use the same idea as in our second treatment of determinism checking earlier, and use $Clunk(F_i(P))$ and $Clunk(G_i(P))$ then we can control the order in which the processes perform actions. For example we can create a process in which

- Any non-*clunk* action immediately before the kth *clunk* is the jth action of process i, where (i, j) is the kth member of an enumeration of $\mathbb{N} \times \mathbb{N}$. If there is no such action then this process will never be permitted to communicate again. This allows us to compare the constituent refinements action-by-action without having an infinite alphabet.
- The ith process is only started up by some appropriate recursive structure once it is actually allowed

to perform actions. This will mean that at any time there are only finitely many processes running in parallel.

The details are left to the interested reader.

Returning briefly to the issue of failures, recall that one of our difficulties in combining failures specifications is that each failure only contains a single refusal set. Just as with the move from finite to infinite traces, it seems that there is a richer model where things may be easier. This is a model called the *refusal testing* model [Muk93] (based on earlier work by Phillips [Phi87]) where refusal sets are recorded throughout the trace rather than just at the end.

This means that constructions like the second version of binary disjunction will now work over this model (which, since it is more expressive than the failures/divergences model $\mathcal{N}$, can certainly express any predicate that $\mathcal{N}$ can). However, since it is still (of course) the case that refusal information does not contribute to traces or divergence information under any CSP operator, there is no chance that we can perform tricks using divergence like $AllTr_Q(P)$ which tell us anything about refusal sets. Therefore there would be more need than with traces for a variant of the refusal-testing model which as far as the author is aware has never been studied, namely one in which there are infinite traces (with infinitely many refusal sets interspersed).

6. Conclusions

In this paper we have shown, as the author had long suspected, that wide-ranging classes of predicates on CSP could be expressed using refinement checking. We have shown precisely which predicates are expressible using functions satisfying (UC) (uniform continuity).

As ever, answering one question raises others. The previous section has gone some way towards answering the question of which predicates are expressible using more general functions, but leaves quite a lot of gaps to be filled in.

A second issue is to gain more insight into the question of which predicates can be captured using finitary F and G (ones which are finite-state if their arguments are), for these are the ones which are genuinely decidable using FDR. Some recent work of the author on this question is reported in [Ros04].

A natural question to arise from this work is that of whether there is some interesting temporal/modal logic which one could show was entirely captured by refinement checking. This question has already been answered in the affirmative for LTL[13] (expressing properties, in essence, on complete and infinite traces: essentially a type of extended behavioural predicate) in [LMC01]. But that work is very different from the present paper. It encompasses many non-closed predicates and therefore could only be compared with our work in Section 5. It is also more restricted in that it only considers specifications of a single execution path (universally quantified over all such). Essentially an LTL predicate is a behavioural predicate on finite and infinite execution sequences. The type of predicate dealt with in the present paper allows us to express predicates on sets of behaviours belonging to a process rather than just one of them. Note, however, that since this is the nature of models based on individual process executions, there can be no way of detecting exactly when some nondeterministic choice was made, so we are not in the conventional branching-time world either.

Finally, the classes of predicates which are on the one hand refinement closed and on the other distributive have been thrown into focus by this work and the examples discussed above. These should be the subject of future work.

Appendix: Notation

This paper follows the notation of [Ros98], from which most of the following is taken. Details of all the operators and models may be found in [Ros98] (URL given in the bibliography).

[13] Although there the atomic predicates talk only about performance of events, not refusals. To extend it to the latter requires (at least) the refusal testing model.

$\mathbb{N}$	natural numbers ($\{0,1,2,\ldots\}$)
Σ	(Sigma): alphabet of all communications
τ	(tau): the invisible action
$\checkmark$	(tick): the action representing successful termination
A^n	set of all length n sequences over A
A^*	set of all finite sequences over A
A^ω	set of all infinite sequences over A
$\langle\rangle$	the empty sequence
$\langle a_1,\ldots,a_n\rangle$	the sequence containing $a_1,\ldots,\ a_n$ in that order
a^ω	the infinite trace $\langle a,a,a,\ldots\rangle$
$s\hat{\ }t$	concatenation of two sequences
$s\setminus X$	hiding: all members of X deleted from s
$s\upharpoonright X$	restriction: the members of s that are in X
	which share members of X and are disjoint elsewhere.
$s\le t$	($\equiv \exists\, u. s\hat{\ }u = t$) prefix order
$\equiv$	equality between boolean expressions and predicates

Processes:

$STOP$	the process that does nothing
$SKIP$	the process that simply terminates ($\checkmark$) successfully
$Chaos_A$	the most nondeterministic divergence-free process over $A\subseteq\Sigma$
div	the process that does nothing but diverge (equivalent to $\mathbf{div}\sqcap Chaos_\Sigma$ in $\mathcal{N}$)
$\mu\, p.P$	recursion
$a\to P$	prefixing
$?x:A\to P$	prefix choice
$b\&P$	conditional execution, equals "`if` b `then` P `else` $STOP$"
$P\,\Box\, Q$	external choice
$P\sqcap Q,\quad \bigsqcap S$	nondeterministic choice
$P\ {}_A\|_B\ Q$	alphabetised parallel
$P \underset{X}{\|} Q$	generalised parallel: P and Q synchronise on X
$P\ \|\|\|\ Q$	interleaving (unsynchronised) parallel
$P\setminus X$	hiding
$P[\![R]\!]$	renaming (relational)
$P\rhd Q$	'time-out' operator (sliding choice)
$P\bigtriangleup Q$	interrupt
$P[x/y]$	substitution (for a free identifier x)
$P\downarrow n$	The least refined process which behaves identically to P for n steps. In $\mathcal{N}$ it diverges (unless P already has) after every length n trace of P.
$d(P,Q)$	metric distance between P and Q $=\ \inf\{2^{-n}\mid P\downarrow n = Q\downarrow n\}$

Models:

$\mathcal{T}$	traces model
$\mathcal{N}$	failures/divergences model (divergence strict)
$\mathcal{N}^+$	divergence- and $\checkmark$-free portion of $\mathcal{N}$
$\mathcal{F}$	stable failures model
$\mathcal{U}$	failures/divergences/infinite traces model (divergence strict)
$\sqsubseteq$	failures/divergences refinement (reverse containment)
$\sqsubseteq_T$	traces refinement
$\sqsubseteq_{\mathcal{U}}$	refinement over $\mathcal{U}$

Acknowledgements

I would like to thank Joy Reed and Jane Sinclair for inspiring this work via their client/server predicates, and Ranko Lazić for his earlier work on determinism. Michael Goldsmith, Joel Ouaknine, Gavin Lowe and Christie Bolton provided helpful comments on earlier drafts. The clarity of the paper was greatly improved by comments from anonymous referees. The work reported here was funded by grants from ONR and EPSRC.

References

[FDR] Formal Systems (Europe) Ltd., *Failures-Divergence Refinement*, User Manual, obtainable from `http://www.formal.demon.co.uk/fdr2manual/index.html`

[Hoa85] Hoare, C.A.R., *Communicating sequential processes*, Prentice Hall, 1985.

[Laz98] Lazić, R.S., *A semantic study of data-independence with applications to the mechanical verification of concurrent systems*, Oxford University D.Phil thesis, 1998.

[LMC01] Leuschel, A., Massart, T., and Currie, T., How to make FDR Spin: LTL model checking of CSP using refinement. In Oliviera, J.N. and Zave, P., Eds. *Proceedings of Formal Methods Europe FME'2001*, LNCS 2021(DSSE-TR-2000-10), pages 99-118.

[Muk93] Mukkaram, A., *A refusal testing model for CSP*, Oxford University D.Phil thesis, 1993.

[Phi87] Phillips, I., Refusal testing, *Theoretical Computer Science* **50** pp241–284 (1987).

[RSR04] Reed, J.N., Sinclair, J., and Roscoe, A.W., Responsiveness of inter-operating components, *Formal Aspects of Computing* **16**, 4, pp294–411 (2004).

[Ros91] Roscoe, A.W., Topology, computer science and the mathematics of convergence, in *Topology and category theory in computer science* (Reed, Roscoe and Wachter, *eds*), Oxford University Press, 1991.

[Ros92] Roscoe, A.W., An alternative order for the failures model, *Journal of Logic and Computation* **2**, 5, pp557–577, 1992.

[Ros93] Roscoe, A.W., Unbounded nondeterminism in CSP, Journal of Logic and Computation, **3**, pp131–172 (1993).

[Ros95] Roscoe, A.W., CSP and determinism in security modelling, *Proceedings of 1995 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1995.

[Ros98] Roscoe, A.W., *The theory and practice of concurrency*, Prentice-Hall International, 1998. *This reference contains almost all necessary background on CSP. It, like the rest of the author's papers listed here other than [Ros03], can be viewed or down-loaded via* `http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/pubs.html`.

[Ros03] Roscoe, A.W., On the expressiveness of CSP refinement checking (Draft version), *Proceedings of AVoCS'03*, (Southampton University Technical Report).

[Ros04] Roscoe, A.W., Finitary refinement checks for infinitary specifications, *Proceedings of CPA 2004* (IOS Press).