

Refinement-oriented probability for CSP

Carroll Morgan, Annabelle McIver, Karen Seidel and JW Sanders¹

Abstract. Jones and Plotkin give a general construction for forming a probabilistic powerdomain over any directed-complete partial order [Jon90, JP89]. We apply their technique to the failures/divergences semantic model for Communicating Sequential Processes [Hoa85].

The resulting probabilistic model supports a new binary operator, probabilistic choice, and retains all operators of CSP including its two existing forms of choice. An advantage of using the general construction is that it is easy to see which CSP identities remain true in the probabilistic model. A surprising consequence however is that probabilistic choice distributes through all other operators; such algebraic mobility means that the syntactic position of the choice operator gives little information about when the choice actually must occur. That in turn leads to some interesting interaction between probability and nondeterminism.

A simple communications protocol is used to illustrate the probabilistic algebra, and several suggestions are made for accommodating and controlling nondeterminism when probability is present.

Keywords: Probability, concurrency, nondeterminism, refinement, CSP.

1. Introduction

The specific aim of this paper is to add probabilistic choice to Hoare's CSP [Hoa85] — to show how the semantics of that can be defined, and to investigate the resulting process algebra.

More generally we are attempting to incorporate probability into a rigorous development method. Although we focus on CSP, the results may be more widely applicable due largely to the work of Claire Jones: in [Jon90] she presents

Correspondence and offprint requests to: Carroll Morgan, Programming Research Group, Wolfson Building, Parks Road, Oxford OX1 3QD. Electronic mail may be sent to {carroll, anabel, karen, jeff}@comlab.ox.ac.uk.

¹ All authors are members of the Programming Research Group; McIver and Seidel are supported by the EPSRC.

a method for constructing a probabilistic space from an arbitrary directed-complete partial order. By applying that to the semantic space for CSP, we preserve at no cost the notions of testing and observation that its failures-divergences order provides.

In CSP a process is tested by observing a ‘failure’ of it — by interacting with it in a way that forces deadlock — and the process passes the test only if that failure is allowed by its specification. It satisfies the specification if it would pass every conceivable test; it does not satisfy the specification if there is even one test that it does not pass.

In observing probabilistic behaviour we are more tolerant, speaking of a specification’s being satisfied with probability at least p : we mean that in a series of T tests we expect the process to pass at least $p \times T$ of them. The tests themselves are as before.

Note that we do not observe probabilistic choices directly, but only their effects: a coin may be flipped well before any decision is based on the result. That view is a consequence of the general construction employed, which (we will show) allows probability to be distributed through all other operators: in particular, the algebra allows a normal form (for finite processes) in which all probabilistic choices are made first.

Is that view justified? Consider ‘scratch cards’ whose silvered windows may reveal prizes when rubbed out. One way to implement them is to place the prize with a certain distribution on every card: in that case the probabilistic choice is made as late as possible, by the customer ‘at run time’ just before he rubs. In particular, it occurs *after* he has chosen the card. But a second way would be to print two kinds of card, one kind with the prize under every window and the other kind with no prizes at all: the probabilistic choice is then made ‘in the factory’, as early as possible and in particular *before* the customer chooses the card.

Limited to the test ‘rub out a window’, one per card, the customer would notice no difference between the two systems.

In Sections 2–7 we detail the application of Jones’ technique to CSP, yielding ‘PCSP’, and explore the resulting structure: of those, Sec. 3 is a largely informal discussion of the consequences; and Secs. 4–5 describe features of CSP that allow some specialisation beyond the generality of [Jon90].

Section 8 introduces explicit probabilistic choice, and shows that it distributes through all other operators. Sections 9–10 discuss the extent to which the algebra of CSP is retained by PCSP, and in particular the interaction of non-determinism and probability.

Sec. 11 presents an example, a recursive process based on the ‘Stop-and-Wait protocol’ [YL92], and Sec. 12 treats recursion generally in PCSP.

Sections 13–14 evaluate the work and compare it with other approaches.

A glossary of definitions *etc.* is given in Appendix A; We use the syntax

$$(Qs: S \mid range \cdot expr)$$

for Q -quantification (for example Q as \forall , \exists , λ or set comprehension) of the values $expr$ formed when s ranges over those elements of S satisfying $range$. Appendix B summarises that and other mathematical notations that might not be standard.

Appendix C summarises the operators of CSP and gives their syntactic binding precedence for this paper.

For convenience we write ‘CSP’ both for the algebra and for its supporting failures/divergences model.

2. Probability spaces as continuous evaluations

We construct PCSP, the space of probabilistic CSP processes, using the general techniques of [Jon90] which we now outline.

An *inductive partial order* (or *ipo*) is a partially ordered set that contains limits for all its directed subsets: thus it is *directed-complete*, but does not necessarily have a least element. The failures-divergences model of CSP is an *ipo* under its refinement order \sqsubseteq (and in fact does have a least element, *CHAOS*).

Jones’ probabilistic powerdomain construction relies on the open sets of the Scott topology generated by the partial order:

Definition 2.1. Given an *ipo* (X, \sqsubseteq) , a subset Y of X is *Scott-open* (or an *element of the Scott topology on (X, \sqsubseteq)*) if both:

1. Y is *up-closed*: if $x \in Y$ and $x \sqsubseteq x'$ then $x' \in Y$ also; and
2. Y is *inaccessible*: if a directed limit $\bigsqcup Z$ lies in Y then one of its elements must — that is, for $Y, Z \subseteq X$ with Z directed,

$$\bigsqcup Z \in Y \quad \text{implies} \quad z \in Y \text{ for some } z \in Z.$$

□

An important feature of the Scott topology is that the \sqsubseteq -continuous functions between two *ipo*’s (those that preserve directed limits) are the same as the topologically continuous functions determined by their respective Scott topologies (those whose inverses preserve openness).

Functions from the Scott-open sets of an *ipo* (X, \sqsubseteq) to the closed interval $[0, 1]$ of real numbers are called *continuous evaluations* [Jon90, pp. 50,66] if they satisfy the probabilistically motivated conditions below:

Definition 2.2. Given an *ipo* (X, \sqsubseteq) , its Scott topology \mathcal{S} and function $\mathbf{f}: \mathcal{S} \rightarrow [0, 1]$, we say that \mathbf{f} is a *continuous evaluation* if it satisfies these properties:

1. (*co-*) *strictness*: $\mathbf{f}(\emptyset) = 0$ and $\mathbf{f}(X) = 1$;
2. *monotonicity*: for any $Y, Z \in \mathcal{S}$,

$$\text{if } Y \subseteq Z \quad \text{then} \quad \mathbf{f}(Y) \leq \mathbf{f}(Z) ;$$

3. *modularity*: for any $Y, Z \in \mathcal{S}$,

$$\mathbf{f}(Y \cup Z) = \mathbf{f}(Y) + \mathbf{f}(Z) - \mathbf{f}(Y \cap Z) ; \text{ and}$$

4. *continuity*: for \mathcal{Y} a \sqsubseteq -directed subset of \mathcal{S} ,

$$\mathbf{f}(\bigsqcup \mathcal{Y}) = (\bigsqcup \mathcal{Y} : \mathbf{f}(Y)) .$$

□

Our probabilistic space is now defined as follows.

Definition 2.3. The space of probabilistic processes PCSP is the set of continuous evaluations over the *ipo* (CSP, \sqsubseteq), the failures-divergences model with its refinement order. □

<u>Set of standard processes</u>	<u>Probability of containing <i>Flip</i></u>
$\{\}$	0
$\{STOP\}$	0
$\{h \rightarrow STOP\}$	0.5
$\{t \rightarrow STOP\}$	0.5
$\{h \rightarrow STOP, t \rightarrow STOP\}$	1

Fig. 1. Some characteristics of the process *Flip*

Thus a probabilistic process is a function over the Scott-open sets of (standard) processes, giving for each the probability that the (probabilistic) process lies within it.

Strictness prevents (even) probabilistic processes from inhabiting the empty set, and co-strictness prevents their evading membership of the set CSP of all processes. Its second conjunct is not imposed by [Jon90], thus we define a subset of the continuous evaluations there. But that subset is closed under our operators (Secs. 6–8), and in any case is equivalent to Jones’ construction over CSP with its least element *CHAOS* removed.²

Monotonicity reflects that expanding a set increases the probability that a given probabilistic process lies within it. Modularity is an analogue of the familiar rule for measures of finite unions of sets. Finally, continuity allows an evaluation over a directed union to be determined by its value over the elements.

3. Intuitive aspects of PCSP

Def. 2.3 determines our view of probabilistic concurrency: we are less interested in how a process ‘makes decisions’ than in what the process ‘is’. Rather than think ‘the probabilistic process *Flip*, defined

$$Flip := h \rightarrow STOP \oplus_{0.5} t \rightarrow STOP, \quad (1)$$

chooses initially between the events h and t with probability 0.5’, instead we think ‘there is a 0.5 probability that *Flip* is the process $h \rightarrow STOP$, and a 0.5 probability that it is $t \rightarrow STOP$ ’.

Fig. 1 gives some characteristics of *Flip* in those terms: the probability 0 in the second row, for example, shows that *Flip* must participate in some event (it cannot be *STOP*); the fifth row shows that event guaranteed (with probability 1) to be either h or t . From Def. 2.3 we know that *Flip* is a function from Scott-open subsets of CSP to probabilities; thus Fig. 1 tabulates *Flip* over part of its domain.

In Sections 4 and 5 to come, we see that we can restrict that domain to the Scott-open ‘finitary cones’, those sets of the form

$$\{P: CSP \mid F \sqsubseteq P\}, \quad (2)$$

where \sqsubseteq is the CSP refinement order and F is some ‘finite’ CSP process that must eventually diverge. (A precise definition of finite is given in Def. 5.2.) We abbreviate the expression (2) by $F\uparrow$ (regardless of whether F is finite).

² We thank one of the referees for the second observation.

The probability that a process A lies in $F\uparrow$ is by definition the probability that A refines F (that $F \sqsubseteq A$): if we regard F as a specification, we are thus asking for the probability that A satisfies it. A practical reason for finite specifications is illustrated by the following example.

Let $Spec_k$ be the process over events s (succeed) and f (fail) that never performs f more than k times consecutively:

$$\begin{aligned} Spec_k &:= Spec'_k(0) \\ Spec'_k(n) &:= \begin{array}{ll} s \rightarrow Spec'_k(0) & \text{if } n < k \\ \square f \rightarrow Spec'_k(n+1) & \end{array} \\ &:= s \rightarrow Spec'_k(0) & \text{if } n = k. \end{aligned}$$

Process $Spec_k$ is a specification, and the larger k is the more likely it is to be met by a process performing f and s events at random.

A probabilistic attempt at implementing $Spec_k$ is

$$Imp_p := s \rightarrow Imp_p \oplus_p f \rightarrow Imp_p,$$

the process which performs s or f repeatedly, with a probability p of the former on each occasion: the closer p is to 1, the more ‘reliable’ is Imp_p .

Now the probability that Imp_p satisfies $Spec_k$, that $Spec_k \sqsubseteq Imp_p$, is the probability that Imp_p lies in the set $Spec_k\uparrow$. We write that as

$$Spec_k \sqsubseteq Imp_p,$$

reading it ‘the probability that $Spec_k$ is refined by Imp_p ’. In fact no matter how large k (though still finite), or how close p to 1 (though not equal to it), we have

$$(Spec_k \sqsubseteq Imp_p) = 0,$$

showing that ‘does Imp_p refine $Spec_k$ ’ is not an interesting question to ask about p and k . No matter how tolerant the specification (large k), and no matter how reliable the implementation (p close to 1), specification $Spec_k$ is guaranteed (with probability 1) to be violated by Imp_p if we wait long enough.

Thus $Spec_k$ is too demanding — a more interesting (and reasonable) specification would be a finite one, say $Spec_k^n$, that behaved like $Spec_k$ for its first n events but diverged thereafter. The value of n would represent the expected (required/desired) lifetime of the implementation, in terms of the number of events it is to engage in, and $Spec_k^n \sqsubseteq Imp_p$ would then concern p , k and n , giving the probability of conformance over a period fixed (by n) in advance. After n events, the specification allows any behaviour in an implementation: divergence is refined by anything.

4. Algebraic *ipo*’s and partial joins

In this and the next section we show that two properties of CSP (beyond its being an *ipo*) imply jointly that continuous evaluations are determined by their values on finitary cones, so justifying our interest in (F, \sqsubseteq) .

The first property is that CSP is algebraic:

Definition 4.1. Given an *ipo* (X, \sqsubseteq) , we say that an element x of X is *ipo-finite* if the cone $x\uparrow$ is Scott-open. Equivalently, x is *ipo-finite* iff it is inaccessible: if whenever $x \sqsubseteq \bigsqcup \mathcal{Y}$ we have $x \sqsubseteq y$ for some y in directed \mathcal{Y} . \square

Definition 4.2. An *ipo* (X, \sqsubseteq) is *algebraic* if, for every element x of X , there is a directed subset $Y \subseteq X$ of *ipo*-finite elements such that $x = \bigsqcup Y$. \square

The second property is that of having partial joins — that if two elements have an upper bound then they have a least upper bound:

Definition 4.3. Two elements x, x' of an *ipo* (X, \sqsubseteq) are said to be *consistent* if there is an element y of X with both $x \sqsubseteq y$ and $x' \sqsubseteq y$. \square

Definition 4.4. An *ipo* (X, \sqsubseteq) is said to have *partial joins* if every pair of consistent elements has a least upper bound. For consistent x, x' we write that (unique) least upper bound $x \sqcup x'$. \square

It is easy to show that *ipo*-finiteness is preserved by \sqcup :

Lemma 4.5. If x, x' in X are *ipo*-finite, then $x \sqcup x'$ is *ipo*-finite if it exists. \square

We now turn to our main result for this section: that being algebraic and having partial joins allows us to restrict our attention to finitary cones when considering continuous evaluations. It relies on the fact that the finitary cones form a basis for the Scott topology if the *ipo* is algebraic:

Lemma 4.6. Let (X, \sqsubseteq) be an algebraic *ipo*, and let Y be an arbitrary Scott-open subset of X (an element of \mathcal{S}). Then Y is a union of finitary cones. \square

Theorem 4.7. Let (X, \sqsubseteq) be an algebraic *ipo* with partial joins, and let \mathbf{f} be a continuous evaluation on it. Then \mathbf{f} is determined by its values on finitary cones, those elements $x \uparrow$ of the Scott topology \mathcal{S} on (X, \sqsubseteq) with x *ipo*-finite.

Proof: Let Y be an arbitrary Scott-open subset of X (an element of \mathcal{S}). Our result would follow immediately from Lem. 4.6 and continuity of \mathbf{f} , if the set of finitary cones within Y were \sqsubseteq -directed — but they are not necessarily. Our approximating subsets will therefore be finite unions of finitary cones within Y . The set of those is indeed \sqsubseteq -directed, and (from Lem. 4.6) has union Y .

Now the value of \mathbf{f} over a finite union of finitary cones is given by modularity, if (X, \sqsubseteq) has partial joins, from its value on (single) finitary cones. For the simple (but representative) case of two cones over *ipo*-finite elements x, x' , we have:

- If $x \uparrow$ and $x' \uparrow$ are disjoint, then

$$\mathbf{f}(x \uparrow \cup x' \uparrow) = \mathbf{f}(x \uparrow) + \mathbf{f}(x' \uparrow)$$

by modularity and strictness.

- If $x \uparrow$ and $x' \uparrow$ are not disjoint, then x, x' are consistent, and in fact $x \uparrow \cap x' \uparrow = (x \sqcup x') \uparrow$. By modularity, we have therefore

$$\mathbf{f}(x \uparrow \cup x' \uparrow) = \mathbf{f}(x \uparrow) + \mathbf{f}(x' \uparrow) - \mathbf{f}((x \sqcup x') \uparrow) .$$

\square

To make use of Thm. 4.7 we must show CSP to have the two properties of its premise. The second is easy: since CSP is closed under arbitrary greatest lower bounds (corresponding to general nondeterministic choice), the least upper bound of two processes P, P' is the least process in the refinement order that refines them both — if one exists.

In the next section we show the first property, that CSP is indeed algebraic.

5. Finite processes

We reconsider the space CSP of standard processes, and define for any natural number n the function $\downarrow n$, so that for P in CSP, the process $P\downarrow n$ ‘behaves like P for the first n events, but then diverges’:

Definition 5.1. For standard process P in CSP over alphabet α , and natural number n , the process $P\downarrow n$ is given by

$$\begin{aligned} failures(P\downarrow n) := & \\ & \{(s, X): failures(P) \mid \#s < n\} \\ \cup & \{s: traces(P); t: \alpha^*; X: \mathbb{P}\alpha \mid \#s = n \cdot (s \smallfrown t, X)\} \end{aligned}$$

and

$$\begin{aligned} divergences(P\downarrow n) := & \\ & divergences(P) \\ \cup & \{s: traces(P); t: \alpha^* \mid \#s = n \cdot s \smallfrown t\} . \end{aligned}$$

□

We can then define a process to be CSP-finite if, for some n , applying $\downarrow n$ makes no difference:

Definition 5.2. For standard process P in CSP, and natural number n , process P is said to be n -finite if $P = P\downarrow n$. (Two CSP processes are equal iff they have the same failures and divergences.)

If there is an n such that P is n -finite, then P is said to be *CSP-finite*. □

(Note that $P\downarrow n \sqsubseteq P$ holds whether P is n -finite or not: it is the other inequality that is important. Note also that $\downarrow n$ is a projection, so that $P\downarrow n$ is CSP-finite for any P and n .)

We now show that our two notions of finiteness — Definitions 4.1 and 5.2 — are equivalent. (App. C contains a brief summary of the CSP model, in particular defining $failures(P)$ and $divergences(P)$.)

Lemma 5.3. Let process F be CSP-finite, and let $\mathcal{P} \subseteq \text{CSP}$ be any directed set of standard processes. Then

$$F \sqsubseteq \bigsqcup \mathcal{P} \quad \text{implies} \quad F \sqsubseteq P \text{ for some } P \in \mathcal{P},$$

and so F is *ipo-finite* as well.

Proof: We must at this point (and from now on) assume that our processes have finite alphabets.

If F is CSP-finite, then it is n -finite for some n ; we consider those pairs (s, X) such that

$$\#s < n \text{ and } (s, X) \notin failures(F) .$$

There are only finitely many of them and, since \mathcal{P} is directed with

$$(\cap P: \mathcal{P} \cdot failures(P)) \subseteq failures(F) ,$$

there must be a P in \mathcal{P} such that none of those failures appears in P . Elements (s, X) of P with $\#s \geq n$ appear in F , because F is n -finite, thus establishing by those two cases that

$$failures(P) \subseteq failures(F) .$$

A similar argument applies for divergences, giving $F \sqsubseteq P$ as required. \square

The final ingredient of the main result for this section is well known:

Lemma 5.4. Every standard CSP process P is the limit of a directed set of (CSP-, hence *ipo*-) finite processes.

Proof: We have easily that $P = (\sqcup n: \mathbb{N} \cdot P \downarrow n)$, and each $P \downarrow n$ is CSP-finite. \square

Lem. 5.4 allows us to conclude the converse of Lem. 5.3, that *ipo*-finite processes are CSP-finite: for if a process is not CSP-finite, then the limit exhibited in Lem. 5.4 shows that it is not inaccessible, and thus not *ipo*-finite.

Hence our two notions of finiteness, CSP- and *ipo*-finite, coincide; we therefore drop the prefix, writing simply ‘finite’.

We have now shown that the *ipo* CSP satisfies the conditions of Thm. 4.7, and thus that any PCSP process A is determined by the values $F \sqsubseteq A$ it takes over finite processes F .

6. Standard processes of PCSP

In this section and the next we populate PCSP with the processes and operators of CSP: the space PCSP contains (an image of) CSP as a subset.

As a mnemonic we use capitals A, B for probabilistic processes, F, G for finite standard processes and P, Q for standard (but not necessarily finite) processes.

Definition 6.1. For any process P in CSP, its image \overline{P} in PCSP is the *point evaluation* determined by

$$F \sqsubseteq \overline{P} := \begin{cases} 1, & \text{if } F \sqsubseteq P \\ 0, & \text{otherwise,} \end{cases}$$

for any finite process F . \square

Informally, one regards the ‘0-or-1’ alternatives as indicating that a standard process P , even if viewed probabilistically, either refines a given F (probability 1) or doesn’t (probability 0).

In fact PCSP itself is a directed-complete partial order [Jon90, p. 66], with its order generated pointwise by the usual order on $[0, 1]$:

Definition 6.2. For probabilistic A, B in PCSP we say that A is *refined by* B , writing $A \sqsubseteq B$, whenever for all Scott-open subsets \mathcal{P} of CSP we have

$$A(\mathcal{P}) \leq B(\mathcal{P}).$$

Limits of directed subsets of PCSP are taken pointwise. \square

Thus A is refined by B if, for any Scott-open \mathcal{P} , process B is at least as likely to inhabit \mathcal{P} as A is. Clearly $A \sqsubseteq B$ implies $F \sqsubseteq A \leq F \sqsubseteq B$, and thus for any finite F that B is at least as likely to refine F as A is. Surprisingly however, the converse does not hold.

That the embedding from CSP to PCSP preserves order is proved in [Jon90, p. 69] — note the ‘if and only if’:

Lemma 6.3. For any processes P, Q in CSP,

$$P \sqsubseteq Q \quad \text{iff} \quad \overline{P} \sqsubseteq \overline{Q}.$$

\square

7. Standard operators of PCSP

CSP processes are constructed with operators that take one or several processes to a single process. For example, the function ‘prefix by event a ’ — which we could write $(a \rightarrow \diamond)$, using \diamond as a placeholder for the argument — is of type $\text{CSP} \rightarrow \text{CSP}$. Similarly, parallel composition \parallel is of type $\text{CSP}^2 \rightarrow \text{CSP}$. We consider only continuous operators, of whatever arity; and all of the usual CSP operators indeed are continuous.

From [Jon90, p. 70] it follows that our construction of PCSP from CSP is the action of a categorical functor. Sec. 6 described the action of the functor on objects: indeed, we could simply define $\text{PCSP} := \overline{\text{CSP}}$, reusing for the functor the overbar symbol we chose to denote embedding of standard processes. If we write $\overline{\diamond}$ for the functor itself, we are now considering the effect of $\overline{\diamond}$ on arrows:

Definition 7.1. Consider *ipo*’s $(X, \sqsubseteq_X), (Y, \sqsubseteq_Y)$ and let $\alpha: X \rightarrow Y$ be a continuous function between them. Then $\overline{\alpha}: \overline{X} \rightarrow \overline{Y}$ is defined

$$\overline{\alpha}(f)(Y') := f(\alpha^{-1}(Y')) ,$$

for all f in \overline{X} and Scott-open Y' in $\mathbb{P}Y$. \square

Thus the probability that $\overline{\alpha}(f)$ assigns to a set Y' is the probability that f assigns to the (inverse) image of Y' through α^{-1} .

The abstract nature of Def. 7.1 is useful when proving properties of embedded functions in general. Given our Thm. 4.7, however, we can reformulate the embedding of standard functions in terms of finitary cones, and even a further simplification is possible in the case that the function is *distributive* (distributes \sqcap):

Theorem 7.2. Let f be a continuous and distributive function in $\text{CSP} \rightarrow \text{CSP}$; then \overline{f} in $\text{PCSP} \rightarrow \text{PCSP}$ satisfies

$$F \sqsubseteq \overline{f}(A) = (\sqcup \text{finite } G: \text{CSP} \mid F \sqsubseteq f(G) \cdot G \sqsubseteq A) ,$$

for all finite F in CSP , and A in PCSP .

Proof:

$$\begin{aligned} & F \sqsubseteq \overline{f}(A) \\ = & \overline{f}(A)(F\uparrow) && \text{definition of } \sqsubseteq \\ = & A(f^{-1}(F\uparrow)) && \text{Def. 7.1} \\ = & A(\sqcup \text{finite } G: \text{CSP} \mid G\uparrow \subseteq f^{-1}(F\uparrow) \cdot G\uparrow) && \text{Lem. 4.6} \\ = & A(\sqcup \text{finite } G: \text{CSP} \mid F \sqsubseteq f(G) \cdot G\uparrow) && \text{definitions } F\uparrow, G\uparrow \\ = & \text{the } G\uparrow\text{'s form a } \sqsubseteq\text{-directed set (see below); } A \text{ continuous} \\ & (\sqcup \text{finite } G: \text{CSP} \mid F \sqsubseteq f(G) \cdot A(G\uparrow)) \\ = & (\sqcup \text{finite } G: \text{CSP} \mid F \sqsubseteq f(G) \cdot G \sqsubseteq A) . && \text{definition of } \sqsubseteq \end{aligned}$$

The reason we argue above that ‘the $G\uparrow$ ’s form a \sqsubseteq -directed set’ (where we could not in the proof of Thm. 4.7) is the special nature of the constraint $F \sqsubseteq f(G)$. We have from the distributivity of f that whenever $F \sqsubseteq f(G_0)$ and $F \sqsubseteq f(G_1)$ then also $F \sqsubseteq f(G_0 \sqcap G_1)$; and both $G_0\uparrow$ and $G_1\uparrow$ are contained by $(G_0 \sqcap G_1)\uparrow$.

Although $G_0 \sqcap G_1$ itself is not necessarily finite³, still there must be a finite G with $F \sqsubseteq f(G)$ and $G \sqsubseteq (G_0 \sqcap G_1)$. \square

Each of the following lemmas is justified by $\overline{\cdot}$'s being a functor [Jon90, Thm. 4.3] on the category of *ipo*'s.

Lemma 7.3. Embedded identity is identity: $\overline{id} = id$. \square

Lemma 7.4. Embedding preserves composition: for continuous f, g in $\text{CSP} \rightarrow \text{CSP}$,

$$\overline{f \circ g} = \overline{f} \circ \overline{g}.$$

\square

Lemma 7.5. Embedding preserves application: for continuous f in $\text{CSP} \rightarrow \text{CSP}$, and P in CSP , we have

$$\overline{f(P)} = \overline{f}(\overline{P}).$$

\square

Lemma 7.6. Embedded functions are continuous.

Proof: It is shown in [Jon90, Thm. 4.3] that the target (category) of the functor $\overline{\cdot}$ has (only) continuous morphisms as arrows. \square

For binary operators, we consider unary functions of type $\text{CSP}^2 \rightarrow \text{CSP}$, to which Def. 7.1 applies: the embedded function is then of type

$$\overline{\text{CSP}^2} \rightarrow \overline{\text{CSP}}.$$

Although the space $\overline{\text{CSP}^2}$ of continuous evaluations over CSP^2 is richer than simply pairs of PCSP processes (which would be $\overline{\text{CSP}^2}$, or equivalently PCSP^2), we treat only those continuous evaluations in $\overline{\text{CSP}^2}$ which are the product of two continuous evaluations in $\overline{\text{CSP}}$, so building in the assumption that probabilistic choices in separate arguments are resolved independently.

Because the special properties of CSP (algebraic, partial joins) carry over to CSP^2 , Thm. 4.7 applies there as well, allowing us (in most cases) to restrict attention to finitary cones rather than considering Scott-open sets more generally. Since CSP is algebraic, each finitary cone in CSP^2 is simply the Cartesian product of a pair of finitary cones in CSP.

Definition 7.7. For finite F, G in CSP and probabilistic A, B in $\overline{\text{CSP}}$, the product (A, B) in $\overline{\text{CSP}^2}$ is given by

$$(A, B)(F\uparrow \times G\uparrow) := A(F\uparrow) \times B(G\uparrow).$$

Since every finitary cone in $\overline{\text{CSP}^2}$ can be written as $F\uparrow \times G\uparrow$ for finite F, G in CSP, Thm. 4.7 then shows that our definition of (A, B) extends uniquely to give its value over arbitrary Scott-open subsets of CSP^2 . \square

The value of $\overline{\alpha}(A, B)$ is thus given by Defs. 7.1 and 7.7 together; a similar argument applies to operations of higher arity.

³ In fact it is, but only because we are working with CSP: *ipo*-finiteness is not in general preserved by \square .

8. Probabilistic choice

We now introduce explicit probability.

For processes A, B , and probability p in the real interval $[0, 1]$, we write $A_p \oplus B$ for the process that is A (behaves like A) with probability p , and is B with probability $1 - p$:

Definition 8.1. For A, B in PCSP and Scott-open subset \mathcal{P} of CSP,

$$(A_p \oplus B)(\mathcal{P}) \quad := \quad p \times A(\mathcal{P}) + (1 - p) \times B(\mathcal{P}) .$$

□

Lemma 8.2. For A, B in PCSP, finite F in CSP and probability p ,

$$F \sqsubseteq A_p \oplus B \quad = \quad p \times (F \sqsubseteq A) + (1 - p) \times (F \sqsubseteq B) .$$

□

The effect of Lem. 8.2 is illustrated by the examples of Sec. 3: we have, eliding the product symbol when convenient,

$$\begin{aligned} h \rightarrow STOP &\sqsubseteq \overline{h \rightarrow STOP}_{0.5 \oplus t \rightarrow STOP} \\ = & \quad 0.5(h \rightarrow STOP \sqsubseteq \overline{h \rightarrow STOP}) && \text{Lem. 8.2} \\ &+ \quad 0.5(h \rightarrow STOP \sqsubseteq \overline{t \rightarrow STOP}) \\ = & \quad 0.5(1) + 0.5(0) && \text{Def. 6.1} \\ = & \quad 0.5 . \end{aligned}$$

We will no longer indicate embedding explicitly in examples (overbar) if it is clear from context.

For $(STOP \sqsubseteq)$ we discover $0.5(0) + 0.5(0)$, and for the fifth case we use

$$(h \rightarrow STOP \sqcap t \rightarrow STOP) \uparrow$$

since $\{h \rightarrow STOP, t \rightarrow STOP\}$ is not a cone. That gives

$$\begin{aligned} h \rightarrow STOP \sqcap t \rightarrow STOP &\sqsubseteq \overline{h \rightarrow STOP}_{0.5 \oplus t \rightarrow STOP} \\ = & \quad 0.5(1) + 0.5(1) \\ = & \quad 1 , \end{aligned}$$

thus justifying Fig. 1 by calculation.

Lem. 8.2 gives $_p \oplus$ the sort of quasi-associativity one would expect, that

$$(A_p \oplus B)_q \oplus C \quad = \quad A_r \oplus (B_s \oplus C)$$

whenever $pq = r$ and $(1 - q) = (1 - r)(1 - s)$, and others similar. Also it is quasi-commutative ($A_p \oplus B = B_{1-p} \oplus A$), idempotent ($A_p \oplus A = A$) and distributes through itself:

$$A_p \oplus (B_q \oplus C) \quad = \quad (A_p \oplus B)_q \oplus (A_p \oplus C) .$$

Our main result, however, is the universal distribution of probabilistic choice through embedded operators; we treat unary operators first.

Theorem 8.3. Probabilistic choice distributes through all embedded unary operators.

Proof: We have, for A, B in PCSP, probability p , Scott-open subset \mathcal{P} of CSP and continuous f in $\text{CSP} \rightarrow \text{CSP}$,

$$\begin{aligned}
& \overline{f}(A \oplus_p B)(\mathcal{P}) \\
= & (A \oplus_p B)(f^{-1}(\mathcal{P})) && \text{Def. 7.1} \\
= & p \times A(f^{-1}(\mathcal{P})) + (1-p) \times B(f^{-1}(\mathcal{P})) && \text{Def. 8.1} \\
= & p \times \overline{f}(A)(\mathcal{P}) + (1-p) \times \overline{f}(B)(\mathcal{P}) && \text{Def. 7.1} \\
= & (\overline{f}(A) \oplus_p \overline{f}(B))(\mathcal{P}) . && \text{Def. 8.1}
\end{aligned}$$

□

Thm. 8.3 reduces our dependence on Lem. 8.2 for explicit calculations. Thus for example

$$\begin{aligned}
& \begin{array}{c} h \rightarrow h \rightarrow \text{STOP} \\ \approx \\ \begin{array}{c} (h \rightarrow (h \rightarrow \text{STOP}_{0.5} \oplus t \rightarrow \text{STOP})) \\ \oplus_{0.5} \quad t \rightarrow (h \rightarrow \text{STOP}_{0.5} \oplus t \rightarrow \text{STOP}) \end{array} \end{array} \\
= & \begin{array}{c} h \rightarrow h \rightarrow \text{STOP} \\ \approx \\ \begin{array}{c} ((h \rightarrow h \rightarrow \text{STOP}_{0.5} \oplus h \rightarrow t \rightarrow \text{STOP})) \\ \oplus_{0.5} \quad (t \rightarrow h \rightarrow \text{STOP}_{0.5} \oplus t \rightarrow t \rightarrow \text{STOP}) \end{array} \end{array} && \text{Thm. 8.3} \\
= & 0.5(0.5(1) + 0.5(0)) + 0.5(0.5(0) + 0.5(0)) && \text{Lem. 8.2} \\
= & 0.25 ,
\end{aligned}$$

so calculating the probability of two heads initially.

Other standard unary CSP operators include hiding and renaming. As an example of more algebraic reasoning, we have for hiding

$$\begin{aligned}
& (a \rightarrow c \rightarrow \text{STOP}_p \oplus b \rightarrow c \rightarrow \text{STOP}) \underline{\text{hide}} \{a, b\} \\
= & \begin{array}{c} (a \rightarrow c \rightarrow \text{STOP}) \underline{\text{hide}} \{a, b\} \\ \oplus_p \quad (b \rightarrow c \rightarrow \text{STOP}) \underline{\text{hide}} \{a, b\} \end{array} && \text{Thm. 8.3} \\
= & c \rightarrow \text{STOP}_p \oplus c \rightarrow \text{STOP} && \text{Lem. 7.5} \\
= & c \rightarrow \text{STOP} . && \text{idempotence of } \oplus_p
\end{aligned}$$

Note that Lem. 7.5 allows us to reason normally in the probabilistic space when dealing only with standard processes and operators.

For distribution of \oplus_p through binary operators, our first step is to consider pair formation (Def. 7.7):

Lemma 8.4. For A, B, C in PCSP, and probability p ,

$$(A \oplus_p B, C) = (A, C) \oplus_p (B, C) .$$

Proof: We consider first only the finitary cones in $\overline{\text{CSP}^2}$: for finite F, G ,

$$\begin{aligned}
= & \begin{array}{c} (A \oplus_p B, C)(F \uparrow \times G \uparrow) \\ (A \oplus_p B)(F \uparrow) \times C(G \uparrow) \end{array} && \text{Def. 7.7} \\
= & \begin{array}{c} p \times A(F \uparrow) \times C(G \uparrow) + (1-p) \times B(F \uparrow) \times C(G \uparrow) \end{array} && \text{Def. 8.1; distribution of addition} \\
= & p \times (A, C)(F \uparrow \times G \uparrow) + (1-p) \times (B, C)(F \uparrow \times G \uparrow) && \text{Def. 7.7}
\end{aligned}$$

$$= (A, C)_{p \oplus} (B, C) . \quad \text{Def. 8.1}$$

To extend the result to arbitrary Scott-open subsets of CSP^2 , we note that the relevant construction in the proof of Thm. 4.7 uses only sums, differences and limits over $[0, 1]$, all of which are linear operators. \square

We now extend Thm. 8.3 to binary operators.

Theorem 8.5. Probabilistic choice distributes through embedded binary operators.

Proof: We show $p \oplus$ distributes through each of the operands of the embedded binary operator $\overline{\otimes}$, whence Thm. 8.3 suffices. For Scott-open subset \mathcal{P} of CSP , and PCSP processes A, B, C ,

$$\begin{aligned} & ((A_{p \oplus} B) \overline{\otimes} C)(\mathcal{P}) \\ = & (A_{p \oplus} B, C)(\otimes^{-1}(\mathcal{P})) && \text{Def. 7.1} \\ = & p \times (A, C)(\otimes^{-1}(\mathcal{P})) + (1 - p) \times (B, C)(\otimes^{-1}(\mathcal{P})) && \text{Lem. 8.4} \\ = & ((A \overline{\otimes} C)_{p \oplus} (B \overline{\otimes} C))(\mathcal{P}) . && \text{Def. 7.1; Def. 8.1} \end{aligned}$$

\square

Thms. 8.3 and 8.5 reveal an important characteristic of our model: since probabilistic choice distributes through all other operators, any non-recursive process can be expressed as a probabilistic choice between finitely many standard processes — one simply distributes all the probabilistic choice to the outside. For example, if P, Q, R, S are embedded standard processes then

$$\begin{aligned} & (P_{0.5 \oplus} Q) \parallel (R_{0.3 \oplus} S) \\ = & (P \parallel (R_{0.3 \oplus} S))_{0.5 \oplus} (Q \parallel (R_{0.3 \oplus} S)) && \text{distributing } 0.5 \oplus \\ = & (P \parallel R_{0.3 \oplus} P \parallel S)_{0.5 \oplus} (Q \parallel R_{0.3 \oplus} Q \parallel S) , && \text{distributing } 0.3 \oplus \end{aligned}$$

revealing a probabilistic combination of standard processes. For ‘nested’ probabilities such as the above, we sometimes use the alternative notation

$$\begin{aligned} P \parallel R & \quad @ 0.15 \\ P \parallel S & \quad @ 0.35 \\ Q \parallel R & \quad @ 0.15 \\ Q \parallel S & \quad @ 0.35 , \end{aligned}$$

indicating a probabilistic ‘dot product’ of the two vectors $\langle P \parallel R, P \parallel S, Q \parallel R, Q \parallel S \rangle$ and $\langle 0.15, 0.35, 0.15, 0.35 \rangle$.

9. Embedding algebraic laws

Lemmas 7.3–7.5 show that embedded standard processes in PCSP satisfy the laws they did in CSP. More interesting, however, is that many of those laws hold for probabilistic processes also.

We regard a law as a relation of equality or refinement between functions, and such relations persist on their extended domain. Take for example the standard law

$$a \rightarrow (P \parallel Q) \quad = \quad (a \rightarrow P) \parallel (a \rightarrow Q) .$$

Each side can be viewed as the application of a function of type $\text{CSP}^2 \rightarrow \text{CSP}$ to a process pair (P, Q) , and so the law asserts equality of the functions:

$$(a \rightarrow \diamond) \circ (\parallel) = (\parallel) \circ ((a \rightarrow \diamond) \times (a \rightarrow \diamond)), \quad (3)$$

where we define $(f \times g)(a, b) := (f(a), g(b))$ for functions f, g .

Extending the law to PCSP thus asserts the embedded equality

$$(\overline{a \rightarrow \diamond}) \circ \overline{\parallel} = \overline{\parallel} \circ (\overline{a \rightarrow \diamond} \times \overline{a \rightarrow \diamond}); \quad (4)$$

and to extract (4) from (3), we argue as follows:

$$\begin{aligned} & \frac{(\parallel) \circ ((a \rightarrow \diamond) \times (a \rightarrow \diamond))}{(\parallel) \circ ((a \rightarrow \diamond) \times (a \rightarrow \diamond))} = \frac{(a \rightarrow \diamond) \circ (\parallel)}{(a \rightarrow \diamond) \circ (\parallel)} \\ \text{hence } & \frac{(\parallel) \circ ((a \rightarrow \diamond) \times (a \rightarrow \diamond))}{\parallel \circ (\overline{a \rightarrow \diamond}) \times (\overline{a \rightarrow \diamond})} = \frac{(a \rightarrow \diamond) \circ (\parallel)}{(a \rightarrow \diamond) \circ (\parallel)} \\ \text{hence } & \frac{\parallel \circ (\overline{a \rightarrow \diamond}) \times (\overline{a \rightarrow \diamond})}{\parallel \circ (\overline{a \rightarrow \diamond} \times \overline{a \rightarrow \diamond})} = \frac{\overline{a \rightarrow \diamond} \circ \parallel}{\overline{a \rightarrow \diamond} \circ \parallel} \quad \text{Lem. 7.4} \\ \text{hence } & \frac{\parallel \circ (\overline{a \rightarrow \diamond}) \times (\overline{a \rightarrow \diamond})}{\parallel \circ (\overline{a \rightarrow \diamond} \times \overline{a \rightarrow \diamond})} = \frac{\overline{a \rightarrow \diamond} \circ \parallel}{\overline{a \rightarrow \diamond} \circ \parallel}. \quad \text{see below} \end{aligned}$$

For the last step, we need the equality of $\overline{(a \rightarrow \diamond) \times (a \rightarrow \diamond)}$ and $\overline{(\overline{a \rightarrow \diamond} \times \overline{a \rightarrow \diamond})}$ — but only when applied to pairs (A, B) of PCSP processes (rather than more general elements of CSP^2), because in laws the functions are applied only to such pairs.

Lemma 9.1. For standard functions f, g and probabilistic processes A, B ,

$$\overline{f \times g}(A, B) = \overline{(\overline{f} \times \overline{g})}(A, B).$$

Proof: From Thm. 4.7 we know that it is sufficient consider just the finitary cones; thus

$$\begin{aligned} & \overline{f \times g}(A, B)(F \uparrow \times G \uparrow) \\ = & (A, B)(f \times g)^{-1}(F \uparrow \times G \uparrow) \quad \text{Def. 7.1} \\ = & (A, B)(f^{-1}(F \uparrow) \times g^{-1}(G \uparrow)) \\ = & A(f^{-1}(F \uparrow)) \times B(g^{-1}(G \uparrow)) \quad \text{Def. 7.7} \\ = & \overline{f}(A)(F \uparrow) \times \overline{g}(B)(G \uparrow) \quad \text{Def. 7.1} \\ = & (\overline{f}(A), \overline{g}(B))(F \uparrow \times G \uparrow) \quad \text{Def. 7.7} \\ = & \overline{(\overline{f} \times \overline{g})}(A, B)(F \uparrow \times G \uparrow). \end{aligned}$$

□

Many of the laws can be reformulated as in the example above, and the proof that they embed is just as straightforward. However we must confine our functions to those constructed from composition of CSP operators and Cartesian products.

Definition 9.2. A function is said to be *simple* in CSP if it is constructed from a composition of only CSP operators and Cartesian products. □

Definition 9.3. A law is said to be *functional* if it is equivalent to a relation of (\sqsubseteq) or $(=)$ between simple functions. □

We now prove the main result, that all functional laws embed:

Theorem 9.4. All functional laws in CSP hold in PCSP also.

Proof: Let $f \sqsubseteq g$ be a relation between functions (representing a functional law). We must convert the embeddings $\overline{f}, \overline{g}$ into simple functions in CSP. Both f, g are simple, and since $\overline{}$ is a functor we know also that $\overline{f}, \overline{g}$ are compositions of embedded constructors and possibly functions of the form $\otimes \circ (h \times k)$ for simple functions h, k and binary operator \otimes . We reason now for all pairs (A, B) in $\overline{\text{CSP}}$:

$$\begin{aligned}
& \overline{\otimes} \circ (h \times k)(A, B) \\
= & \overline{(\otimes \circ h \times k)}(A, B) && \text{Lem. 7.4} \\
= & \overline{(\otimes \circ (\overline{h} \times \overline{k}))}(A, B) . && \text{Lem. 9.1}
\end{aligned}$$

The result follows by induction on the number of Cartesian products in f, g . \square

Thm. 9.4 automatically gives us many embedded laws, some of which are listed in Fig. 2. Its applicability relies on Def. 9.3, that the laws should be functional; the ones that are not are characterised syntactically by the presence of a duplicated argument. For example the idempotence of \sqcap in CSP,

$$P = P \sqcap P ,$$

does not embed (as discussed further in Sec. 10).

Sometimes duplication is avoided by weakening $=$ to \sqsubseteq : for example, in the law

$$P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R)$$

the left-hand side is a simple function but not the right. However the refinements

$$\begin{aligned}
& P \parallel (Q \sqcap R) \sqsubseteq P \parallel Q \\
\text{and } & P \parallel (Q \sqcap R) \sqsubseteq P \parallel R
\end{aligned}$$

have no duplication, and so Thm. 9.4 applies.

Also useful is that any law extends to PCSP when all the duplicated arguments are known to be standard — still allowing probabilities in the non-duplicated arguments. Thm. 9.4 applies because of the following observation:

Lemma 9.5. For any (continuous) binary operator \otimes in $\text{CSP}^2 \rightarrow \text{CSP}$, probabilistic A and standard P ,

$$\overline{P \otimes} A = \overline{(P \otimes \diamond)}(A) .$$

Proof: Writing $P \otimes A$ as $((\otimes) \circ (P, \diamond))(A)$ shows it sufficient to establish $(\overline{P}, \diamond) = \overline{(P, \diamond)}$. That follows, by Thm. 4.7, by considering finitary cones $F \uparrow \times G \uparrow$ and appealing to Def. 7.7. \square

10. Non-determinism

Of the CSP laws that do not hold in PCSP, those involving nondeterminism are conspicuous. Idempotence is one example: consider the processes defined

$$\begin{aligned}
A & := a \rightarrow \text{STOP} \text{ }_{0.5 \oplus} b \rightarrow \text{STOP} \\
\text{and } B & := A \sqcap A .
\end{aligned}$$

Distribution of $_{0.5 \oplus}$ through $\overline{\sqcap}$ (Thm. 8.5) gives

$$\begin{aligned}
& B \\
= & \overline{\sqcap} \begin{array}{l} (a \rightarrow \text{STOP} \text{ }_{0.5 \oplus} b \rightarrow \text{STOP}) \\ (a \rightarrow \text{STOP} \text{ }_{0.5 \oplus} b \rightarrow \text{STOP}) \end{array} \\
= & \text{ }_{0.5 \oplus} \begin{array}{l} a \rightarrow \text{STOP} \sqcap (a \rightarrow \text{STOP} \text{ }_{0.5 \oplus} b \rightarrow \text{STOP}) \\ b \rightarrow \text{STOP} \sqcap (a \rightarrow \text{STOP} \text{ }_{0.5 \oplus} b \rightarrow \text{STOP}) \end{array}
\end{aligned}$$

For probabilistic processes A, B, C , standard P , distinct events a, b and alphabet α ,

$$\begin{aligned}
A \parallel B &= B \parallel A \\
(A \parallel B) \parallel C &= A \parallel (B \parallel C) \\
A \parallel STOP &= STOP, \text{ provided } A \text{ does not diverge} \\
A \parallel RUN &= A, \text{ provided } A \text{ does not diverge} \\
A \parallel CHAOS &= CHAOS \\
(a \rightarrow A) \parallel (a \rightarrow B) &= a \rightarrow (A \parallel B), \text{ provided } a \text{ is synchronised} \\
(a \rightarrow A) \parallel (b \rightarrow B) &= STOP, \text{ provided } a, b \text{ are synchronised} \\
A \parallel B &= B \parallel A \\
A \sqcap B &= B \sqcap A \\
(A \sqcap B) \sqcap C &= A \sqcap (B \sqcap C) \\
(A \parallel B) \parallel C &= A \parallel (B \parallel C) \\
A \parallel STOP &= A \\
A \parallel (B \sqcap C) &\sqsubseteq A \parallel B \\
P \parallel (B \sqcap C) &= (P \parallel B) \sqcap (P \parallel C) \\
A \sqcap (B \parallel C) &\sqsubseteq (A \sqcap B) \parallel C \\
P \sqcap (B \parallel C) &= (P \sqcap B) \parallel (P \sqcap C) \\
(A \sqcap B) \parallel C &\sqsubseteq A \parallel C \\
(A \sqcap B) \parallel P &= (A \parallel P) \sqcap (B \parallel P) \\
(A \sqcap B) \underline{\text{hide}} \alpha &= (A \underline{\text{hide}} \alpha) \sqcap B \underline{\text{hide}} \alpha \\
(a \rightarrow A) \underline{\text{hide}} \alpha &= a \rightarrow (A \underline{\text{hide}} \alpha), \text{ provided } a \notin \alpha \\
(a \rightarrow A) \underline{\text{hide}} \alpha &= A \underline{\text{hide}} \alpha, \text{ provided } a \in \alpha \\
(a \rightarrow A \parallel b \rightarrow B) \underline{\text{hide}} \{a\} &\sqsubseteq A \underline{\text{hide}} \{a\} \parallel b \rightarrow B \underline{\text{hide}} \{a\} \\
(a \rightarrow A \parallel b \rightarrow B) \underline{\text{hide}} \{a\} &\sqsubseteq A \underline{\text{hide}} \{a\} \\
(a \rightarrow P \parallel b \rightarrow B) \underline{\text{hide}} \{a\} &= P \underline{\text{hide}} \{a\} \sqcap (P \underline{\text{hide}} \{a\} \parallel b \rightarrow B \underline{\text{hide}} \{a\})
\end{aligned}$$

(We omit overbars over the operators.)

Fig. 2. Some embedded laws.

$$\begin{aligned}
&= \\
&\quad 0.5 \oplus \quad (a \rightarrow STOP \sqcap a \rightarrow STOP) \\
&\quad 0.5 \oplus \quad (a \rightarrow STOP \sqcap b \rightarrow STOP) \\
&\quad 0.5 \oplus \quad (b \rightarrow STOP \sqcap a \rightarrow STOP) \\
&\quad 0.5 \oplus \quad (b \rightarrow STOP \sqcap b \rightarrow STOP) \\
&= \\
&\quad a \rightarrow STOP \quad @ 0.25 \\
&\quad a \rightarrow STOP \sqcap b \rightarrow STOP \quad @ 0.5 \\
&\quad b \rightarrow STOP \quad @ 0.25,
\end{aligned}$$

using idempotence of $\overline{\parallel}$ on standard operands and commutativity of $\overline{\parallel}$ in general. It is clear that B and A are not equal, and thus that the universal distributivity of $p \oplus$ implies non-idempotence of $\overline{\parallel}$.

Intuitively, we note that process $A \overline{\parallel} A$ becomes $a \rightarrow STOP$ only when the probabilistic choices in both its instances of A are resolved that way. Assuming those choices are independent, that gives a probability of 0.25 — not the 0.5 that $a \rightarrow STOP \sqsubseteq A$ would yield on its own. In fact the 0.25 is the maximum ‘guaranteed’ probability of $a \rightarrow STOP$, since depending on the resolution of nondeterminism in the middle branch (@0.5) the observed frequency could be as high as 0.75. Complementary remarks apply to $b \rightarrow STOP$.

Mathematically we have the following characteristics of $\overline{\parallel}$:

Lemma 10.1. For A, B in PCSP, and \mathcal{P} a Scott-open subset of CSP,

$$(A \bar{\sqcap} B)(\mathcal{P}) \leq A(\mathcal{P}) \times B(\mathcal{P}) .$$

Proof:

$$\begin{aligned} & (A \bar{\sqcap} B)(\mathcal{P}) \\ = & (A, B)(\sqcap^{-1}(\mathcal{P})) && \text{Def. 7.1} \\ \leq & (A, B)(\mathcal{P} \times \mathcal{P}) && \text{see below} \\ = & A(\mathcal{P}) \times B(\mathcal{P}) . && \text{Def. 7.7} \end{aligned}$$

The inequality follows from the fact that $\sqcap^{-1}(\mathcal{P})$ is contained in $\mathcal{P} \times \mathcal{P}$, a special property of \sqcap . It is proved as follows:

$$\begin{aligned} & \sqcap^{-1}(\mathcal{P}) \\ = & \{(P, Q): \text{CSP}^2 \mid P \sqcap Q \in \mathcal{P}\} \\ \subseteq & \{(P, Q): \text{CSP}^2 \mid P \in \mathcal{P} \wedge Q \in \mathcal{P}\} && \text{Def. 2.1: } \mathcal{P} \text{ up-closed} \\ = & \mathcal{P} \times \mathcal{P} . \end{aligned}$$

□

In the special case that \mathcal{P} is a finitary cone $F\uparrow$, the idempotence of \sqcap allows us to strengthen Lem. 10.1 to equality:

Lemma 10.2. For finite F and probabilistic A, B ,

$$F \sqsubseteq A \bar{\sqcap} B = (F \sqsubseteq A) \times (F \sqsubseteq B) .$$

Proof: In the proof of Lem. 10.1 we strengthen the \subseteq -inequality to an equality, given that \mathcal{P} is $F\uparrow$, a finitary cone. For

$$\begin{aligned} & P \sqcap Q \in F\uparrow \\ \text{iff} & F \sqsubseteq P \sqcap Q \\ \text{iff} & F \sqsubseteq P \wedge F \sqsubseteq Q \\ \text{iff} & P \in F\uparrow \wedge Q \in F\uparrow . \end{aligned}$$

□

Lem. 10.2 exhibits the characteristic multiplication of probabilities when choices are resolved independently, and from it we can see how the idempotence of $\bar{\sqcap}$ is lost:

$$F \sqsubseteq A \bar{\sqcap} A = (F \sqsubseteq A) \times (F \sqsubseteq A) \leq F \sqsubseteq A .$$

In general the inequality will be strict when A is probabilistic (when $F \sqsubseteq A$ is neither 0 nor 1).

Probability distributions give more information than $\bar{\sqcap}$, in the sense that in repeated tests a distribution will reveal itself whereas $\bar{\sqcap}$ can ‘do as it pleases’: thus non-deterministic choice can be refined to any probabilistic choice.

Lemma 10.3. For any PCSP processes A, B , and probability p ,

$$A \bar{\sqcap} B \sqsubseteq A_p \oplus B .$$

Proof: We show directly that refinement holds. Let \mathcal{P} be a Scott-open subset of CSP; then

$$\leq \frac{(A \bar{\sqcap} B)(\mathcal{P})}{A(\mathcal{P}) \times B(\mathcal{P})} \quad \text{Lem. 10.1}$$

$$\begin{aligned} &\leq && p \times A(\mathcal{P}) + (1 - p) \times B(\mathcal{P}) && \text{arithmetic, for any } p \\ &= && (A \oplus_p B)(\mathcal{P}) . && \text{Lem. 8.1} \end{aligned}$$

□

In fact, replacing demonic nondeterminism by a choice with fixed probability radically changes its properties: not only is it no longer demonic, it becomes independent of all other choices.

The surprising (mis)behaviour of non-determinism follows inescapably from the distribution of \oplus_p through it, and for example the equality

$$A \sqcap (B \oplus_p C) = A \sqcap B \oplus_p A \sqcap C$$

shows how little one can reason in terms of ‘when a probabilistic choice occurs’. Although on the left the \oplus_p ‘occurs after’ the \sqcap (operationally speaking), on the right it ‘occurs before’: and yet by \oplus_p -distribution those two processes are equal. Thus even on the left we can be sure only that the probabilistic choice occurs no later than the first event of B or C — in particular, we cannot be sure it did not occur before the \sqcap , as the right-hand side shows.

In general, we cannot assume that a probabilistic choice is decided just before it is carried out — the ‘coin may have been flipped’ much earlier, and intervening nondeterminism can take advantage of the result.

11. Example: a simple protocol

Our example is taken from [YL92]: it is a probabilistic version of the Stop-and-Wait protocol [Tan88], which achieves flow control and overcomes the unreliability of a lossy communication link by the simple expedient of (re)transmitting until an acknowledgement is received.

Fig. 3 shows its structure: the sender S and the medium M synchronise on *send* and *timeout* events, the sender and receiver R on *ack*, and the receiver and medium on *receive*. Only *accept* and *deliver* represent interaction with the environment.

We assume that the medium loses messages with constant probability p , and for simplicity we abstract from message values:

$$\begin{aligned} M &:= \text{send} \rightarrow M' \\ M' &:= \text{timeout} \rightarrow M \oplus_p \text{receive} \rightarrow M . \end{aligned}$$

Note that although there is only one occurrence of \oplus_p syntactically in M as given above, unfolding that recursion reveals a distinct choice for each message processed. As discussed in the introduction, although we cannot determine externally at what point those choices are made — whether as each message is received, or perhaps as early as when the medium was constructed — we can still be sure that the choice for each message is made with probability p , independently of the choice for all other messages.

The sender accepts a message from the environment, sends it to the medium, and if it is lost keeps resending it until acknowledged:

$$\begin{aligned} S &:= \text{accept} \rightarrow S' \\ S' &:= \text{send} \rightarrow S'' \\ S'' &:= \text{timeout} \rightarrow S' \parallel \text{ack} \rightarrow S . \end{aligned}$$

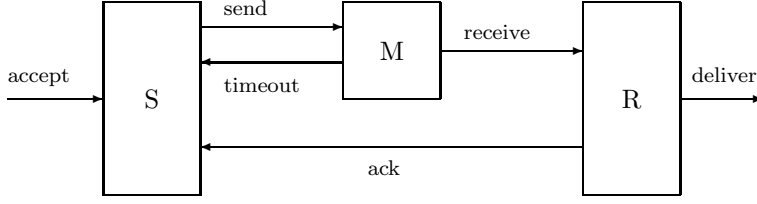


Fig. 3. Structure of the Stop-and-Wait protocol

The receiver receives a message from the medium, then delivers it to the environment and acknowledges it to the sender. For simplicity we assume that acknowledgements are transmitted reliably:

$$R := receive \rightarrow deliver \rightarrow ack \rightarrow R .$$

Taking γ to be the set of internal events $\{send, timeout, receive, ack\}$, our aim is to show that if we hide γ then the resulting system behaves as a simple buffer of capacity 1: a reliable protocol which delivers every message it accepts. That is, we show that

$$(S \parallel M \parallel R) \underline{\text{hide}} \gamma = (\mu B \cdot accept \rightarrow deliver \rightarrow B) . \quad (5)$$

To do so we use several algebraic laws. The first two concern the synchronisation of events in parallel composition and the third concerns hiding. All three are valid because they are functional laws of CSP which, according to Thm. 9.4, hold also in PCSP.

In each of these laws probabilistic A, B are arbitrary; and we use here an explicit indication of the synchronising alphabet, although above and subsequently we have omitted the alphabet subscripts to avoid clutter.

Law 11.1. For alphabet β , events $a \notin \beta$ and $b \in \beta$,

$$a \rightarrow A \parallel_{\beta} b \rightarrow B = a \rightarrow (A \parallel_{\beta} b \rightarrow B) .$$

□

Law 11.2. For alphabet β and event $a \in \beta$,

$$a \rightarrow A \parallel_{\beta} a \rightarrow B = a \rightarrow (A \parallel_{\beta} B) .$$

□

Law 11.3. For alphabet β and event a ,

$$(a \rightarrow A) \underline{\text{hide}} \beta = \begin{cases} A \underline{\text{hide}} \beta & \text{if } a \in \beta \\ a \rightarrow (A \underline{\text{hide}} \beta) & \text{otherwise.} \end{cases}$$

□

We now proceed with our examination of the protocol. First we have

$$= \begin{array}{l} S \parallel M \parallel R \\ \text{accept} \rightarrow (S \parallel M \parallel R) . \end{array} \quad \text{Law 11.1 twice}$$

Then, continuing with the body,

$$\begin{aligned}
& S' \parallel M \parallel R \\
= & \text{send} \rightarrow (S'' \parallel M' \parallel R) && \text{Law 11.2; Law 11.1} \\
= & \text{send} \rightarrow (\quad S'' \parallel (\text{timeout} \rightarrow M) \parallel R && \text{Lem. 8.5} \\
& \quad p \oplus S'' \parallel (\text{receive} \rightarrow M) \parallel R) \\
= & \text{send} \rightarrow (\quad \text{timeout} \rightarrow (S' \parallel M \parallel R) && \text{Law 11.1; Law 11.2} \\
& \quad p \oplus \text{receive} \rightarrow \text{deliver} \rightarrow \text{ack} \rightarrow (S \parallel M \parallel R)) .
\end{aligned}$$

Finally, hiding γ we have both

$$= \frac{(S \parallel M \parallel R) \text{ hide } \gamma}{\text{accept} \rightarrow (S' \parallel M \parallel R) \text{ hide } \gamma} , \quad \text{Law 11.3}$$

and

$$\begin{aligned}
& (S' \parallel M \parallel R) \text{ hide } \gamma \\
= & (S' \parallel M \parallel R) \text{ hide } \gamma \quad p \oplus \text{deliver} \rightarrow (S \parallel M \parallel R) \text{ hide } \gamma . && \text{Law 11.3; Lem. 8.3}
\end{aligned}$$

To deal with the circularity in the last equation, over $(S' \parallel M \parallel R) \text{ hide } \gamma$, we consider the more general

$$P = P \quad p \oplus (a \rightarrow Q) .$$

For any such P, Q , we have

$$F \sqsubseteq P = p \times (F \sqsubseteq P) + (1 - p) \times (F \sqsubseteq a \rightarrow Q)$$

for all F , which by simple arithmetic gives $(F \sqsubseteq P) = (F \sqsubseteq a \rightarrow Q)$ provided p is not 1. Since F is arbitrary, we conclude that $P = a \rightarrow Q$; specifically, we conclude

$$(S \parallel M \parallel R) \text{ hide } \gamma = \text{accept} \rightarrow \text{deliver} \rightarrow (S \parallel M \parallel R) \text{ hide } \gamma .$$

It will be shown in the next section that the equation above has a unique solution, thus defining $(S \parallel M \parallel R) \text{ hide } \gamma$ recursively.

12. Fixed points: least, and unique

The existence of least fixed points in PCSP is guaranteed by the fact that PCSP is an *ipo* and from our use of continuous functions only:

Lemma 12.1. All process-to-process functions written using the standard CSP operators and $p \oplus$ are continuous.

Proof: Lem. 7.6 deals with the standard operators, and continuity is preserved by composition. That leaves only $p \oplus$, for which the result is an easy consequence of the continuity of addition and multiplication over the reals. \square

Because continuous functions over *ipo*'s have least fixed points, Lem. 12.1 justifies recursive definitions such as of this coin-flipping process (like *Flip* but repeating):

$$\text{Flips} := h \rightarrow \text{Flips} \quad 0.5 \oplus t \rightarrow \text{Flips} . \quad (6)$$

For processes like those of Sec. 11, however, one must make the usual extension of Lem. 12.1 to the appropriate product space PCSP^n : then the definition of

$$A \quad := \quad \mathcal{F}(A) ,$$

assumes A is a vector of processes in PCSP^n and \mathcal{F} is in $\text{PCSP}^n \rightarrow \text{PCSP}^n$.

As in CSP, it is helpful in PCSP to be able to recognise situations in which the fixed point of \mathcal{F} is unique, because then from $A = \mathcal{F}(A)$ one can conclude $A = \mu\mathcal{F}$. The technique for that, in (standard) CSP, is to identify those functions \mathcal{F} that are ‘constructive’ in the following sense:

A function $\mathcal{F}: \text{CSP} \rightarrow \text{CSP}$ is said to be constructive if, for all (standard) processes P and natural numbers n ,

$$\mathcal{F}(P \downarrow n) \downarrow (n+1) \quad = \quad \mathcal{F}(P) \downarrow (n+1) .$$

Informally, if \mathcal{F} is constructive then the first $n+1$ events of $\mathcal{F}(P)$ are determined by the first n events of P (and Thm. 12.6 below shows that such constructive \mathcal{F} ’s have unique fixed points).

Since $\downarrow n$ is itself a continuous function (in $\text{CSP} \rightarrow \text{CSP}$), we define constructive for PCSP in the same way:

Definition 12.2. A function $\mathcal{F}: \text{PCSP} \rightarrow \text{PCSP}$ is said to be *constructive* if, for all probabilistic A and naturals n ,

$$\mathcal{F}(A \overline{\downarrow n}) \overline{\downarrow (n+1)} \quad = \quad \mathcal{F}(A) \overline{\downarrow (n+1)} .$$

□

We follow our usual convention of dropping the overbar (from $\overline{\downarrow n}$ in this case) when it’s clear that embedding is meant.

We see easily that standard constructive functions remain constructive when embedded — writing Def. 12.2 as a composition of functions

$$(\downarrow (n+1)) \circ \mathcal{F} \circ (\downarrow n) \quad = \quad (\downarrow (n+1)) \circ \mathcal{F} ,$$

we appeal to Lem. 7.4. In addition, we have that ${}_p\oplus$ preserves being constructive:

Lemma 12.3. Let \mathcal{F}, \mathcal{G} be constructive on $\text{PCSP} \rightarrow \text{PCSP}$. Then

$$(\lambda A \cdot \mathcal{F}(A) \quad {}_p\oplus \quad \mathcal{G}(A))$$

is constructive also.

Proof:

$$\begin{aligned} & (\lambda A \cdot \mathcal{F}(A) \quad {}_p\oplus \quad \mathcal{G}(A))(A \downarrow n) \downarrow (n+1) \\ = & (\mathcal{F}(A \downarrow n) \quad {}_p\oplus \quad \mathcal{G}(A \downarrow n)) \downarrow (n+1) \\ = & \mathcal{F}(A \downarrow n) \downarrow (n+1) \quad {}_p\oplus \quad \mathcal{G}(A \downarrow n) \downarrow (n+1) && \text{Thm. 8.3} \\ = & \mathcal{F}(A) \downarrow (n+1) \quad {}_p\oplus \quad \mathcal{G}(A) \downarrow (n+1) && \mathcal{F}, \mathcal{G} \text{ constructive} \\ = & (\lambda A \cdot \mathcal{F}(A) \quad {}_p\oplus \quad \mathcal{G}(A))(A) \downarrow (n+1) . && \text{Thm. 8.3} \end{aligned}$$

□

Lem. 12.3 thus demonstrates that the definition (6) of *Flips* is a constructive one.

To show that probabilistic constructive functions have unique fixed points, we examine $\downarrow n$ more closely:

Lemma 12.4. For arbitrary finite F , let n be such that F is n -finite. Then, for any probabilistic A ,

$$F \sqsubset A \quad = \quad F \sqsubset A \downarrow n .$$

Proof:

$$\begin{aligned}
& F \sqsubseteq_{\approx} A \downarrow n \\
= & (\sqcup \text{ finite } G: \text{CSP} \mid F \sqsubseteq G \downarrow n \cdot G \sqsubseteq_{\approx} A) && \text{Thm. 7.2} \\
= & (\sqcup \text{ finite } G: \text{CSP} \mid F \downarrow n \sqsubseteq G \downarrow n \cdot G \sqsubseteq_{\approx} A) && F \text{ is } n\text{-finite} \\
\geq & (\sqcup \text{ finite } G: \text{CSP} \mid F \sqsubseteq G \cdot G \sqsubseteq_{\approx} A) && (\downarrow n) \text{ monotonic} \\
= & F \sqsubseteq_{\approx} A && \text{Def. 2.2} \\
\geq & F \sqsubseteq_{\approx} A \downarrow n . && (\downarrow n) \sqsubseteq id; \text{Thm. 9.4; Def. 6.2}
\end{aligned}$$

□

Lem. 12.4 shows that $A \downarrow n$ and A are indistinguishable by \sqsubseteq_{\approx} with respect to n -finite processes. But in fact it gives us the analogue of the limit used in Lem. 5.4:

Lemma 12.5. For probabilistic A ,

$$A = (\sqcup n: \mathbb{N} \cdot A \downarrow n) .$$

Proof: Take any finite F , without loss of generality n -finite. Then

$$\begin{aligned}
& F \sqsubseteq_{\approx} A \\
\geq & F \sqsubseteq_{\approx} (\sqcup n: \mathbb{N} \cdot A \downarrow n) && A \downarrow n \sqsubseteq A \text{ for all } n; \text{Def. 6.2} \\
\geq & F \sqsubseteq_{\approx} A \downarrow n && \text{Def. 6.2} \\
= & F \sqsubseteq_{\approx} A . && F \text{ is } n\text{-finite; Lem. 12.4}
\end{aligned}$$

□

Lem. 12.5 is all we need to show that the fixed point of constructive \mathcal{F} is unique, and the proof is exactly as for the standard case:

Theorem 12.6. Let $\mathcal{F}: \text{PCSP} \rightarrow \text{PCSP}$ be constructive. Then \mathcal{F} has exactly one fixed point.

Proof: That \mathcal{F} has at least one fixed point follows from its being continuous, and from PCSP's being a complete partial order. Thus let A and B be any two fixed points of \mathcal{F} ; then $A \downarrow 0 = \text{CHAOS} = B \downarrow 0$, and

$$\begin{aligned}
& A \downarrow (n+1) = B \downarrow (n+1) \\
\text{iff} & \mathcal{F}(A) \downarrow (n+1) = \mathcal{F}(B) \downarrow (n+1) && A, B \text{ fixed points} \\
\text{iff} & \mathcal{F}(A \downarrow n) \downarrow (n+1) = \mathcal{F}(B \downarrow n) \downarrow (n+1) && \mathcal{F} \text{ constructive} \\
\text{if} & A \downarrow n = B \downarrow n ,
\end{aligned}$$

so showing by induction that $A \downarrow n = B \downarrow n$ for all n . That $A = B$ now follows from Lem. 12.5. □

Thm. 12.6 completes the example of Sec. 11, showing that from

$$(S \parallel M \parallel R) \underline{\text{hide}} \gamma = \text{accept} \rightarrow \text{deliver} \rightarrow (S \parallel M \parallel R) \underline{\text{hide}} \gamma$$

we can indeed conclude

$$(S \parallel M \parallel R) \underline{\text{hide}} \gamma = (\mu B \cdot \text{accept} \rightarrow \text{deliver} \rightarrow B) .$$

13. Problems and prospects

Further study of examples shows that the general embedding principle of Thm. 9.4 is not really generous enough: it is necessary to recover still more laws from standard CSP.⁴

13.1. General choice: state

A probabilistic buffer over values 0, 1 might be written

$$PB \quad := \quad (in?x \rightarrow (out!x \rightarrow PB \oplus_p out!\bar{x} \rightarrow PB)) ,$$

where \bar{x} denotes the complement $1 - x$ of x : the process PB appears to have probability p of transmitting each successive input without corruption. From standard CSP however we know that the right-hand side abbreviates

$$\begin{array}{l} in.0 \rightarrow (out!0 \rightarrow PB \oplus_p out!1 \rightarrow PB) \\ \parallel \quad in.1 \rightarrow (out!1 \rightarrow PB \oplus_p out!0 \rightarrow PB) , \end{array}$$

whence distribution of \oplus_p gives us a four-branch probabilistic choice:

$$\begin{array}{l} in.0 \rightarrow out!0 \rightarrow PB \parallel in.1 \rightarrow out!1 \rightarrow PB \quad @ p^2 \\ in.0 \rightarrow out!0 \rightarrow PB \parallel in.1 \rightarrow out!0 \rightarrow PB \quad @ p\bar{p} \\ in.0 \rightarrow out!1 \rightarrow PB \parallel in.1 \rightarrow out!1 \rightarrow PB \quad @ \bar{p}p \\ in.0 \rightarrow out!1 \rightarrow PB \parallel in.1 \rightarrow out!0 \rightarrow PB \quad @ \bar{p}^2 . \end{array}$$

The probability of correct transmission on the first step is then

$$(in?x \rightarrow out!x \rightarrow CHAOS) \quad \sqsubseteq \quad PB ,$$

which is clearly p^2 rather than the p we expected.

The reason for PB 's behaviour is the duplication inherent in the expansion

$$(in.0 \rightarrow \dots) \parallel (in.1 \rightarrow \dots) .$$

Written that way, the process might be implemented (say in hardware) as two independent subsystems — one for dealing with 0's, the other for dealing with 1's — each with its own probability \bar{p} of failure.

To express (if we wish) that there is no such duplication, we abandon the usual encoding of $(in?x \rightarrow \dots)$, giving its meaning instead by expanding the semantics of CSP to CSP_s , including a limited form of state. That state (more an environment) is a map from a finite number of variable names to a finite number of values: the restriction to finiteness ensures that CSP_s remains algebraic.

The value accepted by an input $(in?x \rightarrow P)$ is bound to x in the state passed on to P ; then $(in?x \rightarrow \diamond)$ becomes a unary (continuous) function in $CSP_s \rightarrow CSP_s$, and \oplus_p distributes through it. That distribution gives us

$$PB \quad = \quad \oplus_p \left(\begin{array}{l} (in?x \rightarrow out!x \rightarrow PB) \\ (in?x \rightarrow out!\bar{x} \rightarrow PB) \end{array} \right) ,$$

with now the expected probability p of correct transmission on each step.

⁴ More complete explanations of the following techniques are given in [MMSS95, MMSS].

13.2. Duplication: *where* clauses

The CSP law for ‘unravelling’ parallel composition is

$$a \rightarrow P \parallel b \rightarrow Q = \begin{array}{l} a \rightarrow (P \parallel b \rightarrow Q) \\ \parallel \\ b \rightarrow (a \rightarrow P \parallel Q) \end{array}, \quad (7)$$

in the case that neither a nor b belongs to the events synchronised by the composition. Yet that law does not embed into the probabilistic space: both P and Q appear twice on the right-hand side.

The loss of that law (and others) prevents an implementor from using two copies of a device when only one is intended. We saw above how that might convert an expected probability of p into a delivered (and unsatisfactory) probability of p^2 ; indeed failure of (7) is consistent with loss of idempotence for $\overline{\parallel}$, if $\{a, b\}$ is hidden on both sides.

To distribute \parallel in the probabilistic space, we adapt the standard law (7) by imposing a condition on its right-hand side, that the duplication is only apparent and is not to be implemented that way:

$$a \rightarrow A \parallel b \rightarrow B = \begin{array}{l} a \rightarrow (P \parallel b \rightarrow Q) \\ \parallel \\ b \rightarrow (a \rightarrow P \parallel Q) \end{array} \\ \text{where } P, Q := A, B .$$

Informally, the where-clause locates A, B in one position syntactically (hence without duplication), indicating by the bound variables P, Q the points at which A, B are ‘used’. Algebraically, the bound variables can be considered standard processes for manipulation of the where-body, making duplication there irrelevant. Operationally, an implementor supplies a single A and a single B , arranging for the activation of the appropriate one of them when the bound variable P or Q is reached.

The extended algebra allows many more laws to be embedded, provided a suitable where-clause is introduced first: such introduction (or subsequent elimination) may be done whenever the bound variable occurs only once, and the bound variables may then be considered standard for manipulation within the where-body.

In [MMSS95] are two examples of state- and where-based reasoning, applied to chains of probabilistic and nondeterministic buffers.

13.3. Indifferent nondeterminism

The unpleasant properties of $\overline{\parallel}$ include its lack of idempotence and its failure to distribute as widely as its standard version; algebraically, that is due to its distribution of ${}_p\oplus$. Yet $\overline{\parallel}$ seems to be unavoidable: it is easily generated by hiding, and thus it distributes ${}_p\oplus$ if hiding and prefixing do.

If a ‘well behaved’ nondeterminism is needed then it must be added, since $\overline{\parallel}$ seems inescapable. Calling the postulated operator *indifferent* nondeterminism (as opposed to ‘demonic’), we introduce it by a Smyth-like construction similar to that which imposes nondeterminism on an otherwise deterministic system [Smy89]. The resulting space NDCSP comprises those subsets of PCSP which are up-closed and convex: convexity of Smyth-set \mathcal{S} means that $A {}_p\oplus B \in \mathcal{S}$ whenever $A, B \in \mathcal{S}$. Convexity preserves the property

if $A \sqsubseteq B$ and $A \sqsubseteq C$ then $A \sqsubseteq B \oplus_p C$,

present in PCSP, and allows indifferent (as well as demonic) nondeterminism to be refined by \oplus_p .

The resulting extended construction can be shown to be a complete partial order (closed under taking directed limits), and embedded functions (over PCSP) remain continuous. The conditions for embedding laws appear to be much the same as before.

The algebraic properties of the new operator — write it \oplus — relate to existing PCSP functions much as \oplus_p relates to standard functions: indifferent nondeterminism distributes through all other operators, including \oplus_p ; and \oplus_p retains its distribution properties except in the presence of \oplus , just as $\overline{\sqcap}$ retained its distribution properties except in the presence of \oplus_p (and now \oplus also). Furthermore, indifferent nondeterminism is idempotent.

But what is indifferent nondeterminism? It cannot take advantage of probabilistic choice in the way demonic nondeterminism can (since \oplus_p does not distribute through it), and it is therefore most simply resolved ‘in the factory’: process $A \oplus B$ is already either A or B when delivered; it does not only later, ‘at runtime’, resolve the nondeterminism.

In the light of our early comments about testing and observation (Sec. 1), however, we must show how one would detect the difference between $A \oplus B$ and $A \sqcap B$ in practice (since otherwise we should not distinguish them in theory). The experiment is as follows: consider

$$a \rightarrow STOP \sqcap b \rightarrow STOP \quad \textit{versus} \quad a \rightarrow STOP \oplus b \rightarrow STOP,$$

testing each with respect to either

$$a \rightarrow STOP \quad \textit{or} \quad b \rightarrow STOP$$

with probability 0.5 between the two possible tests — the experimenter flips a coin to decide on each occasion which test to apply. Now the case $\overline{\sqcap}$ might fail every test: being demonic, the choice could resolve to $b \rightarrow STOP$ whenever the experimenter chose $a \rightarrow STOP$, and *vice versa*. But the indifferent choice would reveal a failure rate of 0.5 no matter what distribution of $a \rightarrow STOP$ and $b \rightarrow STOP$ ‘leaves the factory’: whether all $a \rightarrow STOP$, all $b \rightarrow STOP$ or some mixture, the experimenter would be wrong (or right) just half the time in the long run.

14. Comparison with other approaches

We can divide much of the work on probabilistic concurrency into two groups (as does [Low]):

- *external* probability, where the environment offers a set of events, and the process chooses probabilistically between them; and
- *internal* probability, where the process chooses probabilistically which set of events to offer the environment.

The difference is not in ‘who’ makes the probabilistic choice (it is always the process); rather it is in who decides the set from which the choice will be made. Our work belongs to the second group.

In the first group we find the ‘generative’ model of [vGSST90, BBS92], and the ‘extended failures’ model of [Low93]: the first is CCS- [Mil89], the second ACP- [BW90] and the third CSP-based. In the generative model (but using our syntax), a probabilistic external choice

$$a \rightarrow A \text{ }_p\parallel\text{ } b \rightarrow B$$

selects the left branch if the environment offers only a , and similarly the right if b . When both a, b are offered, however, the process itself decides between them, with probability p for the left and $1-p$ for the right branch. Extended failures contains a similar operator. In both cases, when p is 0 or 1 the choice is ‘prioritised’, consistently favouring one of the two branches whenever the environment offers both.

Our membership of the second group is forced upon us: it is difficult to see how ‘construction-time’ probability could depend on ‘runtime’ offers. It is a drawback that we cannot easily accommodate external probabilistic choice as well as internal.

A second dimension of comparison is provided by the notion of equivalence. Ours is that two processes are equal when the test $(F \sqsubseteq)$ yields identical results for all finite F . The $(F \sqsubseteq)$ -equivalence is finer than those based on assigning probabilities to the results of trace-like tests, as in [Pnu85, Low93, Sei92], and the ‘reactive’ and generative models of [vGSST90]; yet it is coarser than the ‘stratified bisimulation’ of [vGSST90] and the ‘probabilistic bisimulation’ of [LS91].

If an equivalence is very coarse (identifying ‘many’ processes), then some operators will not be definable (or, alternatively expressed, in the presence of those operators the equivalence will not be a congruence). Lowe’s model [Low93] does not include nondeterminism; Seidel’s model [Sei92] does not include hiding; and in the ‘broom’ and ‘barb’ semantics of [Pnu85] it is not possible to define CSP-style event renaming (as shown in [Low]). But the coarser the equivalence, the more algebraic laws hold for the operators that remain.

If an equivalence is very fine (identifying ‘few’ processes), then there will be fewer algebraic laws; and there may even be distinctions made in the model that its users would rather ignore. The stratified model [vGSST90] is sensitive to the ‘level’ at which probabilistic choices occur, thus forgoing the quasi-associativity of $p \oplus$; with the probabilistic bisimulation of [LS91], probabilistic choice does not distribute through prefixing $(a \rightarrow \diamond)$.

The ‘testing equivalence’ of [YL92] (generalising [dNH84]) appears to be very close to ours; probabilistic choice distributes through most operators, and (CCS-style) choice is no longer idempotent. Still, prefixing $(a \rightarrow \diamond)$ does not distribute probabilistic choice; but that may however be related to more fundamental differences between CCS and CSP.

A third dimension concerns underspecification and nondeterminism. Some development formalisms (*eg.* [Gro92]) distinguish the two in theory: a customer may ask for any of several alternatives each of which is deterministic, thus underspecifying but still excluding nondeterminism. Many other methods [Dij76] do not make that distinction, reasoning that it is relevant only when repeated testing (or equivalently ‘copying’) is allowed.

Yet testing for probability must involve repetition, as one is looking for distributions — thus the underspecification *versus* nondeterminism distinction may

now unavoidably be with us: having increased our resolution, to observe probabilities, we must accept also other phenomena revealed consequentially. Our indifferent nondeterminism may well be underspecification, and since the testing example of Sec. 13.3 involved only standard processes, we can argue that the distinction is not caused directly by the introduction of probability: the new operator ${}_p\oplus$ is ‘why’ we must observe it, but not ‘how’.

In [BB94], Baeten and Bergstra discuss three choice operators in the context of ACP: ‘static’, ‘collecting’ and ‘dynamic’. There are striking similarities with ${}_p\oplus$, \oplus and $\overline{\parallel}$ respectively.

Static choice is made at some stage during the execution of the process, but possibly well before the occurrence of the first event in the chosen alternative. In that respect it is like our ${}_p\oplus$, which can be decided at an earlier stage than it is carried out. (See the discussion concluding Sec. 10.)

Collecting choice is made before all other choices; thus like \oplus it is made in the factory, or ‘before any action is executed in the system’ [BB94, p.465].

Dynamic choice is made as late as possible — just before the execution of the alternative chosen — and so, like our $\overline{\parallel}$, is able to take advantage of all earlier choices.

Thus the two triples of choice operators (static, collecting, dynamic) and $({}_p\oplus, \oplus, \overline{\parallel})$ appear to have similar algebraic properties with respect to each other and to standard operators.

The number of possibilities for choice is interesting, yet puzzling — we have four, including \parallel . Thus we reserve judgement on the utility of \oplus , but it is informative to place it in the context of the other operators:

$$A \overline{\parallel} B \quad \sqsubseteq \quad \begin{cases} A \oplus B \\ A \parallel B \end{cases} \quad \sqsubseteq \quad A {}_p\oplus B$$

Note however that neither \oplus nor ${}_p\oplus$ refines to \parallel in general.

Acknowledgements

We thank Paul Gardiner for his energetic and relentless skepticism, and the referees for their very careful reading and useful suggestions.

References

- [BB94] J.C.M. Baeten and J.A. Bergstra. Process algebra with partial choice. In *CONCUR '94*, volume 836 of *LNCS*, pages 465–480. Springer Verlag, 1994.
- [BBS92] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatising probabilistic processes: ACP with generative probabilities. In *CONCUR '92*, volume 630 of *LNCS*, pages 472–485. Springer Verlag, 1992.
- [BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.
- [Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice Hall International, Englewood Cliffs, N.J., 1976.
- [dNH84] M. de Nicola and M. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34, 1984.
- [Gro92] RAISE Language Group. *The RAISE Specification Language*. Prentice-Hall, 1992.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
- [Jon90] C. Jones. Probabilistic nondeterminism. Monograph ECS-LFCS-90-105, Edinburgh University, 1990. (Ph.D. Thesis).

- [JP89] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the IEEE 4th Annual Symposium on Logic in Computer Science*, pages 186–195, Los Alamitos, Calif., 1989. Computer Society Press.
- [Low] G. Lowe. Representing nondeterministic and probabilistic behaviour in reactive processes. *Formal Aspects of Computing*. To appear.
- [Low93] G. Lowe. Probabilities and priorities in timed CSP. Technical Monograph PRG-111, Oxford University Computing Laboratory, 1993. (DPhil Thesis).
- [LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
- [Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
- [MMSS] A.K. McIver, C.C. Morgan, K. Seidel, and J.W. Sanders. A power construction for nondeterminism over probability. In preparation.
- [MMSS95] C.C. Morgan, A.K. McIver, K. Seidel, and J.W. Sanders. Argument duplication in probabilistic CSP. Technical Report PRG-TR-11-95, Programming Research Group, April 1995. Available at [?, key MMSS94a].
- [Pnu85] A. Pnueli. Linear and branching structure in the semantics and logics of reactive systems. In *Proceedings of 12th International Colloquium on Automata, Languages and Programming*. Springer Verlag, LNCS 458, 1985.
- [Sei92] K. Seidel. Probabilistic communicating processes. Technical Monograph PRG-102, Oxford University, 1992. (DPhil Thesis).
- [Smy89] M.B. Smyth. Power domains and predicate transformers: a topological view. In *Automata, Languages and Programming 10th Colloquium, Barcelona, Spain*, volume 298 of *LNCS*, pages 662–675. Springer Verlag, 1989.
- [Tan88] A.S. Tanenbaum. *Computer Networks*. Prentice-Hall, second edition, 1988.
- [vGSST90] R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C. Tofts. Reactive, generative and stratified models of probabilistic processes. In *IEEE Symposium on Logic in Computer Science, Philadelphia, PA, USA*, June 1990.
- [YL92] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In R.J Linn and M. Ümit Uyar, editors, *Proceedings of 12th IFIP International Symposium on Protocol Specification, Testing and Verification, Florida, USA*, pages 47–61, 1992.

A. Summary of definitions *etc.*

Def. 2.1	Scott-open sets
Def. 2.2	Continuous evaluations
Def. 2.3	The probabilistic space PCSP
Def. 4.1	Finite elements in an <i>ipo</i>
Def. 4.2	Algebraic <i>ipo</i>
Def. 4.3	Consistent elements
Def. 4.4	Partial join
Lem. 4.5	Finiteness is preserved by join
Lem. 4.6	Scott-open sets are unions of cones
Thm. 4.7	Continuous evaluations determined by their values on cones
Def. 5.1	<i>CHAOS</i> after n steps: $\downarrow n$
Def. 5.2	CSP-finiteness
Lem. 5.3	CSP-finite implies <i>ipo</i> -finite
Lem. 5.4	CSP is algebraic
Def. 6.1	Embedding for processes
Def. 6.2	Refinement order in PCSP
Lem. 6.3	Order is preserved by embedding
Def. 7.1	Embedding for unary functions
Thm. 7.2	Embedding for unary functions on cones
Lem. 7.3	Identity is preserved
Lem. 7.4	Composition is preserved

Lem. 7.5	Application is preserved
Lem. 7.6	Embedded functions are continuous
Def. 7.7	Product of continuous evaluations
Def. 8.1	Probabilistic choice
Lem. 8.2	Probabilistic choice on cones
Thm. 8.3	Probabilistic choice distributes through unary operators
Lem. 8.4	Probabilistic choice distributes through pairing
Thm. 8.5	Probabilistic choice distributes through binary operators
Lem. 9.1	Embedded Cartesian product of functions
Def. 9.2	Simple functions
Def. 9.3	Functional laws
Thm. 9.4	Functional laws embed
Lem. 9.5	Binary operator with standard operand
Lem. 10.1	Embedded nondeterministic choice
Lem. 10.2	Embedded nondeterministic choice on cones
Lem. 10.3	Nondeterministic refines to probabilistic choice
Law 11.1	Partially synchronised parallel composition
Law 11.2	Synchronised parallel composition
Law 11.3	Hiding
Lem. 12.1	Probabilistic choice is continuous
Def. 12.2	Constructive functions
Lem. 12.3	Probabilistic choice is constructive-preserving
Lem. 12.4	Finite cones for finite processes
Lem. 12.5	Probabilistic processes are limits of their $\downarrow n$ approximates
Thm. 12.6	Constructive fixed points are unique

B. Mathematical notation

$lhs := rhs$	Define lhs to be rhs .
$\{s: S \mid range\}$	The set of elements s of set S that satisfy $range$.
$F \uparrow$	The cone on F .
$F \sqsubseteq A$	The probability that F is refined by A .
$\mathbb{P}S$	The set of all subsets of S .
$\{s: S \cdot expr\}$	The set of values $expr$ formed as s ranges over S .
$\{s: S \mid range \cdot expr\}$	The set of values $expr$ formed as s ranges over those elements of S that satisfy $range$.
$(\cap s: S \cdot expr)$	The intersection of sets $expr$ as s ranges over S .
$(\cup s: S \mid range \cdot expr)$	The union of sets $expr$ as s ranges over elements of S satisfying $range$.
$(\sqcup s: S \mid range \cdot expr)$	The least upper bound of values $expr$ as s ranges over elements of S satisfying $range$.
\diamond	Place-holder for indicating the argument position in a function.
$P \downarrow n$	Process P restricted to its first n events, diverging thereafter.

For Sec. 5 and App. C, CSP-specific:

$\#s$	The length of trace (sequence) s .
-------	--------------------------------------

α^*	The set of all finite traces over alphabet (set of events) α .
$s \frown t$	The concatenation of traces s and t .
$s \Downarrow \beta$	The trace remaining when all events of s not in β are removed.

C. Summary of CSP operators

Following is a list of CSP and PCSP operators, some of which are introduced in this paper. They are given in order of syntactic binding power, from strongest binding to weakest.

For the (standard) CSP operators we give definitions in terms of the semantic model [Hoa85]:

A CSP process P over a finite⁵ non-empty set α of (uninterpreted) events, the *alphabet*, comprises three structures.

- The *traces* of P , written $traces(P)$, are finite sequences of events in α in which the process P might engage.
- The *failures* of P , written $failures(P)$, are trace/refusal pairs where a *refusal* is a subset of α . The failure (s, X) , for s in α^* and $X \subseteq \alpha$, indicates that P may perform s and then refuse to engage in any event in X .
- The *divergences* of P , written $divergences(P)$, are traces after which the process may behave chaotically.

Process P is refined by process Q , written $P \sqsubseteq Q$, when $failures(P) \supseteq failures(Q)$ and $divergences(P) \supseteq divergences(Q)$.

There are constraints on the traces, failures and divergences of CSP processes, for example that $traces(P)$ is non-empty and prefix closed for any P ; we refer to [Hoa85] for full details. We shall use, however, that $traces(P)$ is determined by $failures(P)$ —

$$traces(P) = \{s \mid (s, \emptyset) \in failures(P)\}$$

— so that below we need define only failures and divergences for each operator. Note that we use the set comprehension syntax of this paper (App. B).

chaos Process *CHAOS* diverges immediately; it is bottom in the CSP refinement order.

$$\begin{aligned} failures(CHAO S) &:= \alpha^* \times \mathbb{P}\alpha \\ divergences(CHAO S) &:= \alpha^* \end{aligned}$$

stop Process *STOP* is the process that does nothing: it can engage in no events.

$$\begin{aligned} failures(STOP) &:= \{\langle \rangle\} \times \mathbb{P}\alpha \\ divergences(STOP) &:= \emptyset \end{aligned}$$

hiding Process $A \text{ hide } \beta$, for process A and subset β of α , acts like process A

⁵ That is a restriction we impose in this paper.

except that occurrences of events in β become ‘internal’, no longer visible to an observer.

$$\begin{aligned} \text{failures}(A \text{ hide } \beta) &:= \\ &\{(s, X) \mid (s, X \cup \beta) \in \text{failures}(A) \cdot (s \Downarrow (\alpha - \beta), X)\} \\ &\cup \text{divergences}(A \text{ hide } \beta) \times \mathbb{P}\alpha \\ \text{divergences}(A \text{ hide } \beta) &:= \\ &\{s, t \mid t \in \alpha^* \wedge \Delta(s) \cdot s \Downarrow (\alpha - \beta) \frown t\} \\ \Delta(s) &:= \begin{array}{l} s \in \text{divergences}(A) \\ \vee (\forall n \cdot (\exists u: \beta^* \cdot \#u > n \wedge s \frown u \in \text{traces}(A))) \end{array} \end{aligned}$$

prefixing Process $a \rightarrow A$ performs event a then behaves like process A .

$$\begin{aligned} \text{failures}(a \rightarrow A) &:= \begin{array}{l} \{\langle \rangle\} \times \mathbb{P}(\alpha - \{a\}) \\ \cup \{(s, X): \text{failures}(A) \cdot (\langle a \rangle \frown s, X)\} \end{array} \\ \text{divergences}(a \rightarrow A) &:= \{s: \text{divergences}(A) \cdot \langle a \rangle \frown s\} \end{aligned}$$

external choice Process $A \parallel B$ behaves either like A or like B , where the selection is determined by the environment’s choice of first event.

$$\begin{aligned} \text{failures}(A \parallel B) &:= \begin{array}{l} \{(s, X): \text{failures}(A) \cap \text{failures}(B) \mid s = \langle \rangle\} \\ \cup \{(s, X): \text{failures}(A) \cup \text{failures}(B) \mid s \neq \langle \rangle\} \\ \cup \text{divergences}(A \parallel B) \times \mathbb{P}\alpha \end{array} \\ \text{divergences}(A \parallel B) &:= \text{divergences}(A) \cup \text{divergences}(B) \end{aligned}$$

internal (demonic) choice Process $A \sqcap B$ behaves either like A or like B , but the environment cannot influence the choice between them.

$$\begin{aligned} \text{failures}(A \sqcap B) &:= \text{failures}(A) \cup \text{failures}(B) \\ \text{divergences}(A \sqcap B) &:= \text{divergences}(A) \cup \text{divergences}(B) \end{aligned}$$

parallel composition Process $A \parallel_{\beta} B$ runs processes A and B ‘concurrently’

(interpreted as interleaving), except that events in the set β require participation of both processes simultaneously. The alphabet β is omitted when clear from context.

$$\begin{aligned} \text{failures}(A \parallel_{\beta} B) &:= \\ &\{s: \alpha^*, (t, X): \text{failures}(A), (u, Y): \text{failures}(B) \\ &\quad \mid \Lambda_{\beta}(s, t, u) \cdot (s, X \cup Y)\} \\ &\cup \text{divergences}(A \parallel_{\beta} B) \times \mathbb{P}\alpha \\ \text{divergences}(A \parallel_{\beta} B) &:= \\ &\{s, t: \text{traces}(A), u: \text{traces}(B) \\ &\quad \mid \Lambda_{\beta}(s, t, u) \wedge (t \in \text{divergences}(A) \vee u \in \text{divergences}(B)) \cdot s\} \\ \Lambda_{\beta}(s, \langle \rangle, s) &:= s \Downarrow \beta = \langle \rangle \\ \Lambda_{\beta}(s, s, \langle \rangle) &:= s \Downarrow \beta = \langle \rangle \\ \Lambda_{\beta}(\langle a \rangle \frown s, t, u) &:= \begin{array}{l} (t = \langle a \rangle \frown t' \wedge \Lambda_{\beta}(s, t', u)) \\ \vee (u = \langle a \rangle \frown u' \wedge \Lambda_{\beta}(s, t, u')) \end{array} \quad \text{if } a \notin \beta \end{aligned}$$

$$\Lambda_\beta(\langle a \rangle \frown s, \langle a \rangle \frown t, \langle a \rangle \frown u) := \Lambda_\beta(s, t, u) \quad \text{if } a \in \beta$$

probabilistic choice Process $A \oplus_p B$ behaves like process A with probability p , and like process B with probability $1 - p$.

indifferent choice Process $A \oplus B$ behaves either like process A or like process B ; but the environment cannot influence the choice between them, and that choice is independent of all other probabilistic or indifferent choices.

where application Process $F(P)$ where $P := A$ resolves probabilistic choices in A once only, no matter how many times P may occur syntactically in $F(P)$.