# On CSP and the Algebraic Theory of Effects

Rob van Glabbeek and Gordon Plotkin*

**Abstract** We consider CSP from the point of view of the algebraic theory of effects, which classifies operations as effect *constructors* and effect *deconstructors*; it also provides a link with functional programming, being a refinement of Moggi's seminal monadic point of view. There is a natural algebraic theory of the constructors whose free algebra functor is Moggi's monad; we illustrate this by characterising free and initial algebras in terms of two versions of the stable failures model of CSP, one more general than the other. Deconstructors are dealt with as homomorphisms to (possibly non-free) algebras.

One can view CSP's action and nondeterminism operators as constructors and the rest, such as concealment and concurrency, as deconstructors. Carrying this programme out results in taking deterministic external choice as constructor rather than general external choice. However, binary deconstructors, such as the CSP concurrency operator, provide unresolved difficulties. We conclude by presenting a combination of CSP with Moggi's computational $\lambda$-calculus, in which the operators, including concurrency, are polymorphic. While the paper mainly concerns CSP, it ought to be possible to carry over similar ideas to other process calculi.

Rob van Glabbeek
NICTA, Sydney, Australia
University of New South Wales, Sydney, Australia

Gordon Plotkin
Laboratory for the Foundations of Computer Science, School of Informatics, University of Edinburgh, UK

* This work was done with the support of a Royal Society-Wolfson Award.

## 1 Introduction

We examine Hoare's CSP [BHR84, Hoa85, Ros98] from the point of view of the algebraic theory of effects [PP02, PP04, HPP06, PPr09], a refinement of Moggi's seminal 'monads as notions of computation' [Mog89, Mog91, BHM02]. This is a natural exercise as the algebraic nature of both points to a possibility of commonality. In the algebraic theory of effects operations do not all have the same character. Some are effect *constructors*: they create the effects at hand; some are effect *deconstructors*: they respond to effects created. For example, raising an exception creates an effect—the exception raised—whereas exception-handling responds to effects—exceptions that have been raised. It may therefore be interesting, and even useful, to classify CSP operators as constructors or deconstructors. Considering CSP and the algebraic theory of effects together also raises the possibility of combining CSP with functional programming in a principled way, as Moggi's monadic approach provides a framework for the combination of computational effects with functional programming. More generally, although we mainly consider CSP, a similar exercise could be undertaken for other process calculi as they have a broadly similar algebraic character.

The theory of algebraic effects starts with the observation that effect constructors generally satisfy natural equations, and Moggi's monad $T$ is precisely the free algebra monad for these equations (an exception is the continuations monad which is of a different character). Effect deconstructors are treated as homomorphisms from the free algebra to another algebra, perhaps with the same carrier as the free algebra but with different operations. These operations can be given by combinations of effect constructors and previously defined deconstructors. The situation is much like that of primitive recursive definitions, although we will not present a formal definitional scheme.

We mainly consider that part of CSP containing action, internal and external choice, deadlock, relabelling, concealment, concurrency and interleaving, but not, for example, recursion (we do, albeit briefly, consider the extension with termination and sequencing). The evident constructors are then action prefix, and the two kinds of nondeterminism, internal and external, the latter together with deadlock. The evident deconstructors are relabelling, concealment, concurrency and interleaving. There is, however, a fly in the ointment, as pointed out in [PPr09]. Parallel operators, such as CSP's concurrency and interleaving, are naturally binary, and respond to effects in both arguments. However, the homomorphic approach to deconstructors, as sketched above, applies only to unary deconstructors, although it is possible to extend it to accommodate parameters and simultaneous definitions. Nonetheless, the natural definitions of concurrency and interleaving do not fall within the homomorphic approach, even in the extended sense. This problem has nothing to do with CSP: it applies to all examples of parallelism of which we are aware.

Even worse, when we try to carry out the above analysis for CSP, it seems that the homomorphic approach cannot handle concealment. The difficulty is caused by the fact that concealment does not commute with external choice. Fortunately this difficulty can be overcome by changing the effect constructors: we remove external

choice and action prefix and replace them by the deterministic external choice operator $(a_1 \to P(a_1) \mid \ldots \mid a_n \to P(a_n))$, where the $a_i$ are all different. Binary external choice then becomes a deconstructor.

With that we can carry out the program of analysis, finding only the expected difficulty in dealing with concurrency and interleaving. However, it must be admitted that the *n*-ary operators are somewhat clumsy to work with, and it is at least a priori odd to take binary external choice as a deconstructor. On the other hand, in [Hoa85, Section 1.1.3] Hoare writes:

> The definition of choice can readily be extended to more than two alternatives, e.g.,
>
> $$(x \to P \mid y \to Q \mid \ldots \mid z \to R)$$
>
> Note that the choice symbol | is *not* an operator on processes; it would be syntactically incorrect to write $P \mid Q$, for processes $P$ and $Q$. The reason for this rule is that we want to avoid giving a meaning to
>
> $$(x \to P \mid x \to Q)$$
>
> which appears to offer a choice of first event, but actually fails to do so.

which might be read as offering some support to a treatment which takes deterministic external choice as a primitive (here = constructor), rather than general external choice. On our side, we count it as a strength of the algebraic theory of effects that it classifies effect-specific operations and places constraints on them: that they either belong to the basic theory or must be defined according to a scheme that admits inductive proofs.

Turning to the combination with functional programming, consider Moggi's computational $\lambda$-calculus. Just as one accommodates imperative programming within functional programming by treating commands as expressions of type `unit`, so it is natural to treat our selection of CSP terms as expressions of type `empty` as they do not terminate normally, only in deadlock. For process languages such as ACP [BK85, BK86] which do have the possibility of normal termination, or CSP with such a termination construct, one switches to regarding process terms as expressions of type `unit`, when a sequencing operator is also available.

As we have constructors for every $T(X)$, it is natural to treat them as polymorphic constructs, rather than just as process combinators. For example, one could have a binary construction for internal choice, with typing rule:

$$\frac{M : \sigma \qquad N : \sigma}{M \sqcap N : \sigma}$$

It is natural to continue this theme for the deconstructors, as in:

$$\frac{M : \sigma}{M \backslash a : \sigma} \qquad \frac{M : \sigma \qquad N : \tau}{M \,||\, N : \sigma \times \tau}$$

where the thought behind the last rule is that $M$ and $N$ are evaluated concurrently, terminating normally only if they both do, when the pair of results returned individually by each is returned.

In the case of CSP a functional programming language CSPM incorporating CSP processes has been given by Scattergood [Sca]; it is used by most existing CSP tools including the Failures Divergences Refinement Checker (FDR), see [Ros94]. Scattergood's CPSM differs from our proposal in several respects. Most significantly, processes are not treated on a par with other expressions: in particular they cannot be taken as arguments in functions, and CSP constructors and deconstructors are only available for processes. It remains to be seen if such differences are of practical relevance.

In Section 3 we take deadlock, action, binary internal and external choice as the constructors. We show, in Theorem 3.4, that, with the standard equational theory, the initial algebra is the 'finitary part' of the original Brookes-Hoare-Roscoe failures model [BHR84]; which is known to be isomorphic to the finitary, divergence- and $\checkmark$-free part of the failures/divergences model, as well as the finitary, divergence- and $\checkmark$-free part of the stable failures model, both of which are described in [Ros98]. In Section 4 we go on to consider effect deconstructors, arriving at the difficulty with concealment and illustrating the problems with parallel operators in the (simpler) context of Milner's synchronisation trees. A reader interested in the problem of dealing with parallel operators algebraically need only read this part, together with [PPr09].

We then backtrack in Section 5, making a different choice of constructors, as discussed above, and giving another characterisation of the finitary failures model as an initial algebra in Theorem 5.2. With that, we can carry out our programme, failing only where expected: with the binary deconstructors. In Section 6 we add a zero for the internal choice operator to our algebra; this can be interpreted as divergence in the stable failures model, and permits the introduction of a useful additional deterministic external choice constructor. Armed with this tool, in Section 7, we look at the combination of CSP and functional programming, following the lines hinted at above. In order to give a denotational semantics we need, in Theorem 7.4, to characterise the free algebras rather than just the initial one.

As remarked above, termination and sequencing are accommodated within functional programming via the type `unit`; in Section 7.1 we therefore also give a brief treatment of our fragment of CSP extended with termination and sequencing, modelling it in the free algebra over the one-point set.

The concluding Section 8 contains a brief discussion of the general question of combining process calculi, or parallelism with a global store, with functional programming. The case of CSP considered here is just one example of the many possible such combinations. Throughout this paper we do not consider recursion; this enables us to work within the category of sets. A more complete treatment would deal with recursion working within, say, the category of $\omega$-cpos (i.e., partial orders with lubs of increasing $\omega$-sequences) and continuous functions (i.e., monotone functions preserving lubs of increasing $\omega$-sequences). This is discussed further in Section 8. The appendix gives a short presentation of Moggi's computational $\lambda$-calculus.

## 2 Technical preliminaries

We give a brief sketch of finitary equational theories and their free algebra monads. For a fuller explanation see, e.g., [Bor94, AGM95]. Finitary equational theories Th are derived from a given set of axioms, written using a signature $\Sigma$ consisting of a set of operation symbols $\mathrm{op} : n$, together with their arities $n \geq 0$. One forms terms $t$ from the signature and variables and the axioms then consist of equations $t = u$ between the terms; there is a natural equational logic for deducing consequences of the axioms; and the theory consists of all the equations derivable from the axioms. A *ground* equation is one where both terms are *closed*, meaning that they contain no variables.

For example, we might consider the fragment of CSP with signature $\Box : 2$, $\mathtt{Stop} : 0$ and the following axioms for a semilattice (the first three axioms) with a zero (the last):

| | |
|---|---|
| Associativity | $(x \Box y) \Box z = x \Box (y \Box z)$ |
| Commutativity | $x \Box y = y \Box x$ |
| Idempotence | $x \Box x = x$ |
| Zero | $x \Box \mathtt{Stop} = x$ |

A $\Sigma$-*algebra* is a structure $\mathscr{A} = (X, (\mathrm{op}_{\mathscr{A}} : X^n \to X)_{\mathrm{op}:n \in \Sigma})$; we say that $X$ is the *carrier* of $\mathscr{A}$ and the $\mathrm{op}_{\mathscr{A}}$ are its *operations*. We may omit the subscript on operations when the algebra is understood. When we are thinking of an algebra as an algebra of processes, we may say 'operator' rather than 'operation.' A homomorphism between two algebras is a map between their carriers respecting their operations; we therefore have a category of $\Sigma$-algebras.

Given such a $\Sigma$-algebra, every term $t$ has a *denotation* $[\![t]\!](\rho)$, an element of the carrier, given an assignment $\rho$ of elements of the carrier to every variable; we often confuse terms with their denotation. The algebra *satisfies* an equation $t = u$ if $t$ and $u$ have the same denotation for every such assignment. If $\mathscr{A}$ satisfies all the axioms of a theory Th, it is called a Th-algebra; the Th-algebras form a subcategory of the category of $\Sigma$-algebras. Any equation provable from the axioms of a theory Th is satisfied by any Th-algebra. We say that a theory Th is *(ground) equationally complete* with respect to a Th-algebra if a (ground) equation is provable from Th if, and only if, it is satisfied by the Th-algebra.

Any finitary equational theory Th determines a free algebra monad $T_{\mathrm{Th}}$ on the category of sets, as well as operations

$$\mathrm{op}_X : T_{\mathrm{Th}}(X)^n \to T_{\mathrm{Th}}(X)$$

for any set $X$ and $\mathrm{op} : n \in \Sigma$, such that $(T_{\mathrm{Th}}(X), (\mathrm{op}_X : X^n \to X)_{\mathrm{op}:n \in \Sigma})$ is the free Th-algebra over $X$. Although $T_{\mathrm{Th}}(X)$ is officially just a set, the carrier of the free algebra, we may also use $T_{\mathrm{Th}}(X)$ to denote the free algebra itself. In the above example the monad is the finite powerset monad:

$$\mathscr{F}(X) = \{u \subseteq X \mid u \text{ is finite}\}$$

with $\Box_X$ and $\mathtt{Stop}_X$ being union and the empty set, respectively.

## 3 A first attempt at analysing CSP

We consider the fragment of CSP with deadlock, action prefix, internal and external choice, relabelling and concealment, and concurrency and interleaving. Working over a fixed alphabet $A$ of *actions*, we consider the following operation symbols:

**Deadlock**

$$\mathtt{Stop}:0$$

**Action**

$$a\rightarrow -:1 \qquad (a\in A)$$

**Internal and External Choice**

$$\sqcap,\Box:2$$

**Relabelling and Concealment**

$$f(-),-\backslash a:1$$

for any *relabelling function* $f:A\rightarrow A$ and action $a$. If $A$ is infinite, this makes the syntax infinitary; as that causes us no problems, we do not avoid it.

**Concurrency and Interleaving**

$$||,|||:2$$

The signature of our (first) equational theory CSP$(\Box)$ for CSP only has operation symbols for the subset of these operators which are naturally thought of as constructors, namely deadlock, action and internal and external choice. Its axioms are those given by de Nicola in [DeN85]. They are largely very natural and modular, and are as follows:

- $\Box,\mathtt{Stop}$ is a semilattice with a zero (i.e., the above axioms for a semilattice with a zero).
- $\sqcap$ is a semilattice (i.e., the axioms stating the associativity, commutativity and idempotence of $\sqcap$).
- $\Box$ and $\sqcap$ distribute over each other:

$$x\Box(y\sqcap z)=(x\Box y)\sqcap(x\Box z) \qquad x\sqcap(y\Box z)=(x\sqcap y)\Box(x\sqcap z)$$

- Actions distribute over $\sqcap$:

$$a\rightarrow(x\sqcap y)=a\rightarrow x\sqcap a\rightarrow y$$

and:

$$a \to x \,\Box\, a \to y = a \to x \sqcap a \to y$$

All these axioms are mathematically natural except the last which involves a relationship between three different operators.

We adopt some useful standard notational abbreviations. For $n \geq 1$ we write $\bigsqcap_{i=1}^{n} t_i$ to abbreviate $t_1 \sqcap \ldots \sqcap t_n$, intending $t_1$ when $n = 1$. We assume that parentheses associate to the left; however as $\sqcap$ is associative, the choice does not matter. As $\sqcap$ is a semilattice, we can even index over nonempty finite sets, as in $\bigsqcap_{i\in I} t_i$, assuming some standard ordering of the $t_i$ without repetitions. As $\Box$ is a semilattice with a zero, we can adopt analogous notations $\bigsqcap_{i=1}^{n} t_i$ and $\bigsqcap_{i\in I} t_i$ but now also allowing $n$ to be 0 and $I$ to be $\emptyset$.

As $\sqcap$ is a semilattice we can define a partial order for which it is the greatest lower bound by writing $t \sqsubseteq u$ as an abbreviation for $t \sqcap u = t$; then, as $\Box$ distributes over $\sqcap$, it is monotone with respect to $\sqsubseteq$: that is, if $x \sqsubseteq x'$ and $y \sqsubseteq y'$ then $x \,\Box\, y \sqsubseteq x' \,\Box\, y'$. (We mean all this in a formal sense, for example, that if $t \sqsubseteq u$ and $u \sqsubseteq v$ are provable, so is $t \sqsubseteq v$, etc.) We note the following, which is equivalent to the distributivity of $\sqcap$ over $\Box$, given that $\sqcap$ and $\Box$ are semilattices, and the other distributivity, that $\Box$ distributes over $\sqcap$:

$$x \sqcap (y \,\Box\, z) = x \sqcap (y \,\Box\, z) \sqcap (x \,\Box\, y) \tag{1}$$

The equation can also be written as $x \sqcap (y \,\Box\, z) \sqsubseteq (x \,\Box\, y)$. Using this one can derive another helpful equation:

$$(x \,\Box\, a \to z) \sqcap (y \,\Box\, a \to w) = (x \,\Box\, a \to (z \sqcap w)) \sqcap (y \,\Box\, a \to (z \sqcap w)) \tag{2}$$

We next rehearse the original refusal sets model of CSP, restricted to finite processes without divergence; this provides a convenient context for identifying the initial model of CSP($\Box$) in terms of failures.

A *failure (pair)* is a pair $(w, W)$ with $w \in A^*$ and $W \subseteq_{\text{fin}} A$. For every set $F$ of failure pairs, we define its set of *traces* to be

$$\text{tr}_F = \{w \mid (w, \emptyset) \in F\}$$

and for every $w \in \text{tr}_F$ we define its set of *futures* to be:

$$\text{fut}_F(w) = \{a \mid wa \in \text{tr}_F\}$$

With that a *refusal set* $F$ (aka a *failure set*) is a set of failure pairs, satisfying the following conditions:

1. $\varepsilon \in \text{tr}_F$
2. $wa \in \text{tr}_F \Rightarrow w \in \text{tr}_F$
3. $(w, W) \in F \wedge V \subseteq W \Rightarrow (w, V) \in F$
4. $(w, W) \in F \wedge a \notin \text{fut}_F \Rightarrow (w, W \cup \{a\}) \in F$

A refusal set is *finitary* if its set of traces is finite.

The collection of finitary refusal sets can be turned into a CSP($\Box$)-algebra $\mathscr{R}_f$ by the following standard definitions of the operators:

$$\begin{array}{ll}
\mathtt{Stop}_{\mathscr{R}_f} & = \{(\varepsilon, W) \mid W \subseteq_{\text{fin}} A\} \\
a \rightarrow_{\mathscr{R}_f} F & = \{(\varepsilon, W) \mid a \notin W\} \cup \{(aw, W) \mid (w, W) \in F\} \\
F \sqcap_{\mathscr{R}_f} G & = F \cup G \\
F \,\Box_{\mathscr{R}_f}\, G & = \{(\varepsilon, W) \mid (\varepsilon, W) \in F \cap G\} \cup \{(w, W) \mid w \neq \varepsilon,\ (w, W) \in F \cup G\}
\end{array}$$

The other CSP operation symbols also have standard interpretations over the collection of finitary refusal sets:

$$\begin{array}{ll}
f(F) & = \{(f(w), W) \mid (w, f^{-1}(W) \cap \text{fut}_F(w)) \in F\} \\
F \backslash a & = \{(w \backslash a, W) \mid (w, W \cup \{a\}) \in F\} \\
F \mid\mid G & = \{(w, W \cup V) \mid (w, W) \in F,\ (w, V) \in G\} \\
F \mid\mid\mid G & = \{(w, W) \mid (u, W) \in F,\ (v, W) \in G,\ w \in u \,|\, v\}
\end{array}$$

with the evident action of $f$ on sequences and sets of actions, and where $w \backslash a$ is obtained from $w$ by removing all occurrences of $a$, and where $u \,|\, v$ is the set of interleavings of $u$ and $v$.

**Lemma 3.1.** *Let $F$ be a finitary refusal set. Then for every $w \in \text{tr}_F$ there are $V_1, \ldots, V_n \subseteq \text{fut}_F(w)$, including $\text{fut}_F(w)$, such that $(w, W) \in F$ iff $W \cap V_i = \emptyset$ for some $i \in \{1, \ldots, n\}$.*

*Proof.* The closure conditions imply that $(w, W)$ is in $F$ iff $(w, W \cap \text{fut}_F(w))$ is. Thus we only need to be concerned about pairs $(w, W)$ with $W \subseteq \text{fut}_F(w)$. Now, as $\text{fut}_F(w)$ is finite, for any relevant $(w, W) \in F$, of which there are finitely many, we can take $V$ to be $\text{fut}_F(w) \backslash W$, and we obtain finitely many such sets. As $(w, \emptyset) \in F$, these include $\text{fut}_F(w)$. $\Box$

**Lemma 3.2.** *All finitary refusal sets are definable by closed* CSP($\Box$) *terms.*

*Proof.* Let $F$ be a finitary refusal set. We proceed by induction on the length of the longest trace in $F$. By the previous lemma there are sets $V_1, \ldots, V_n$, including $\text{fut}_F(\varepsilon)$, such that $(\varepsilon, W) \in F$ iff $W \cap V_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. Define $F_a$, for $a \in \text{fut}_F(\varepsilon)$, by:

$$F_a = \{(w, W) \mid (aw, W) \in F\}$$

Then it is not hard to see that each $F_a$ is a finitary refusal set, and that

$$F = \bigsqcap_i \Box_{a \in V_i} a \rightarrow F_a$$

As the longest trace in $F_a$ is strictly shorter than the longest one in $F$, the proof concludes, employing the induction hypothesis. $\Box$

We next recall some material from de Nicola [DeN85]. Let $\mathscr{L}$ be a collection of sets; we say it is *saturated* if whenever $L \subseteq L' \subseteq \bigcup \mathscr{L}$, for $L \in \mathscr{L}$ then $L' \in \mathscr{L}$. Then

a closed CSP($\Box$)-term $t$ is in *normal form* if it is of the form:

$$\bigsqcap_{L\in\mathscr{L}} \Big[\!\Big]_{a\in L} a \to t_a$$

where $\mathscr{L}$ is a finite non-empty saturated collection of finite sets of actions and each term $t_a$ is in normal form. Note that the concept of normal form is defined recursively.

**Proposition 3.3.** CSP($\Box$) *is ground equationally complete with respect to* $\mathscr{R}_f$.

*Proof.* Every term is provably equal in CSP($\Box$) to a term in normal form. For the proof, follow that of Proposition A6 in [DeN85]; alternatively, it is a straightforward induction in which equations (1) and (2) are helpful. Further, it is an immediate consequence of Lemma 4.8 in [DeN85] that if two normal forms have the same denotation in $\mathscr{R}_f$ then they are identical (and Lemma 7.2 below establishes a more general result). The result then follows. □

**Theorem 3.4.** *The finitary refusal sets algebra* $\mathscr{R}_f$ *is the initial* CSP($\Box$) *algebra.*

*Proof.* Let the initial such algebra be I. There is a unique homomorphism $h : \mathrm{I} \to \mathscr{R}_f$. By Lemma 3.2, $h$ is a surjection. By the previous proposition, $\mathscr{R}_f$ is complete for equations between closed terms, and so $h$ is an injection. So $h$ is an isomorphism, completing the proof. □

## 4 Effect deconstructors

In the algebraic theory of effects, the semantics of effect *deconstructors*, such as exception handlers, is given using homomorphisms from free algebras. In this case we are interested in $T_{\mathrm{CSP}(\Box)}(\emptyset)$. This is the initial CSP($\Box$) algebra, $\mathscr{R}_f$, so given a CSP($\Box$) algebra:

$$\mathscr{A} = (T_{\mathrm{CSP}(\Box)}(\emptyset), \sqcap_{\mathscr{A}}, \mathtt{Stop}_{\mathscr{A}}, (a \to_{\mathscr{A}}), \Box_{\mathscr{A}})$$

there is a unique homomorphism:

$$h : \mathscr{R}_f \to \mathscr{A}$$

**Relabelling** We now seek to define $f(-) : T_{\mathrm{CSP}(\Box)}(\emptyset) \to T_{\mathrm{CSP}(\Box)}(\emptyset)$ homomorphically. Define an algebra $Rl$ on $T_{\mathrm{CSP}(\Box)}(\emptyset)$ by putting, for refusal sets $F, G$:

$$\mathtt{Stop}_{Rl} = \mathtt{Stop}_{\mathscr{R}_f}$$

$$(a \to_{Rl} F) = (f(a) \to_{\mathscr{R}_f} F)$$

$$F \sqcap_{Rl} G = F \sqcap_{\mathscr{R}_f} G \qquad F \Box_{Rl} G = F \Box_{\mathscr{R}_f} G$$

One has to verify this gives a CSP($\Box$)-algebra, which amounts to verifying that the two action equations hold, for example that, for all $F, G$:

$$a \rightarrow_{Rl} (F \sqcap_{Rl} G) = (a \rightarrow_{Rl} F) \sqcap_{Rl} (a \rightarrow_{Rl} G)$$

which is equivalent to:

$$f(a) \rightarrow_{\mathscr{R}_f} (F \sqcap_{\mathscr{R}_f} G) = (f(a) \rightarrow_{\mathscr{R}_f} F) \sqcap_{\mathscr{R}_f} (f(a) \rightarrow_{\mathscr{R}_f} G)$$

We therefore have a unique homomorphism

$$\mathscr{R}_f \xrightarrow{h_{Rl}} Rl$$

and so the following equations hold over the algebra $\mathscr{R}_f$:

$$h_{Rl}(\mathtt{Stop}) = \mathtt{Stop}$$

$$h_{Rl}(a \rightarrow F) = f(a) \rightarrow h_{Rl}(F)$$

$$h_{Rl}(F \sqcap G) = h_{Rl}(F) \sqcap h_{Rl}(G) \qquad h_{Rl}(F \,\Box\, G) = h_{Rl}(F) \,\Box\, h_{Rl}(G)$$

Informally one can use these equations to define $h_{Rl}$ by a '*principle of equational recursion*,' but one must remember to verify that the implicit algebra obeys the required equations.

We use $h_{Rl}$ to interpret relabelling. We then immediately recover the familiar CSP laws:

$$f(\mathtt{Stop}) = \mathtt{Stop}$$

$$f(a \rightarrow x) = f(a) \rightarrow f(x)$$

$$f(x \sqcap y) = f(x) \sqcap f(y) \qquad f(x \,\Box\, y) = f(x) \,\Box\, f(y)$$

which we now see to be restatements of the homomorphism of relabelling.

**Concealment** There is a difficulty here. We do not have that

$$(F \,\Box\, G)\backslash a = F\backslash a \,\Box\, G\backslash a$$

but rather have the following two equations (taken from [DeN85]):

$$((a \rightarrow F) \,\Box\, G)\backslash a = F\backslash a \sqcap ((F \,\Box\, G)\backslash a) \tag{3}$$

$$(\Box_{i=1}^{n} a_i F_i)\backslash a = \Box_{i=1}^{n} a_i (F_i\backslash a) \tag{4}$$

where no $a_i$ is $a$. Furthermore, there is no direct definition of concealment via an equational recursion, i.e., there is no suitable choice of algebra, $\Box_{\mathscr{A}}$ etc. For, if there were, we would have:

$$(F \,\square\, G)\backslash a = F\backslash a \,\square_{\mathscr{A}}\, G\backslash a \tag{5}$$

So if $a$ does not occur in any trace of $F'$ or $G'$ we would have:

$$\begin{aligned} F' \,\square_{\mathscr{A}}\, G' &= F'\backslash a \,\square_{\mathscr{A}}\, G'\backslash a \\ &= (F' \,\square\, G')\backslash a \\ &= F' \,\square\, G' \end{aligned}$$

but, returning to equation (5), $a$ certainly does not occur in any trace of $F\backslash a$ or $G\backslash a$ and so we would have:

$$\begin{aligned} (F \,\square\, G)\backslash a &= F\backslash a \,\square_{\mathscr{A}}\, G\backslash a \\ &= F\backslash a \,\square_{\mathscr{R}_f}\, G\backslash a \end{aligned}$$

which is false. It is conceivable that although there is no direct homomorphic definition of concealment, there may be an indirect one where other functions (possibly with parameters—see below) are defined homomorphically and concealment is definable as a combination of those.

### *4.1 Concurrency operators*

Before trying to recover from the difficulty with concealment, we look at a further difficulty, that of accommodating binary deconstructors, particularly parallel operators. We begin with a simple example in a strong bisimulation context, but rather than a concurrency operator in the style of CCS we consider one analogous to CSP's $||$.

We take as signature a unary action prefix, $a.-$, for $a \in A$, a nullary `NIL` and a binary sum $+$. The axioms are that $+$ is a semilattice with zero `NIL`; the initial algebra is then that of finite synchronisation trees $ST$. Every synchronisation tree $\tau$ has a finite depth and can be written as

$$\sum_{i=1}^{n} a_i.\tau_i$$

for some $n \geq 0$, where the $\tau_i$ are also synchronisation trees (of strictly smaller depth), and where no pair $(a_i, \tau_i)$ occurs twice. The order of writing the summands makes no difference to the tree denoted.

One can define a binary synchronisation operator $||$ on synchronisation trees $\tau = \sum_i a_i.\tau_i$ and $\tau' = \sum_j b_j.\tau_j$ by induction on the depth of $\tau$ (or $\tau'$):

$$\tau \,||\, \tau' = \sum_{a_i = b_j} a_i.(\tau_i \,||\, \tau'_j)$$

Looking for an equational recursive definition of $||$, one may try a 'mutual (parametric) equational recursive definition' of $||$ and a certain family $||^a$ with $x, y, z$ varying

over $ST$:

$$\begin{array}{rl} \mathtt{NIL} \mid\mid z & = \mathtt{NIL} \\ (x+y) \mid\mid z & = (x \mid\mid z) + (y \mid\mid z) \\ a.x \mid\mid z & = x \mid\mid^a z \end{array}$$

and

$$\begin{array}{rl} z \mid\mid^a \mathtt{NIL} & = \mathtt{NIL} \\ z \mid\mid^a (x+y) & = (z \mid\mid^a x) + (z \mid\mid^a y) \\ z \mid\mid^a b.x & = \begin{cases} a.(z \mid\mid x) & (\text{if } b = a) \\ \mathtt{NIL} & (\text{if } b \neq a) \end{cases} \end{array}$$

Unfortunately, this definition attempt is not an equational recursion. Mutual (parametric) equational recursions are single ones to an algebra on a product. Here we wish a map: $ST \to ST \times ST$. Informally we would write such clauses as:

$$\langle (x+y) \mid\mid z \;,\; z \mid\mid^a (x+y) \rangle = \langle (x \mid\mid z) + (y \mid\mid z) \;,\; (z \mid\mid^a x) + (z \mid\mid^a y) \rangle$$

with the recursion variables, here $x,y$, on the left for $\mid\mid$ and on the right for $\mid\mid^a$. However:

$$\langle a.x \mid\mid z \;,\; z \mid\mid^a b.x \rangle = \begin{cases} \langle x \mid\mid^a z \;,\; a.(z \mid\mid x) \rangle & (\text{if } b = a) \\ \langle x \mid\mid^a z \;,\; \mathtt{NIL} \rangle & (\text{if } b \neq a) \end{cases}$$

does not respect this discipline: the recursion variable, here $x$, (twice) switches places with the parameter $z$.

We are therefore caught in a dilemma. One can show, by induction on the depth of synchronisation trees, that the above definitions, viewed as equations for $\mid\mid$ and $\mid\mid^a$ have a unique solution: the expected synchronisation operator $\mid\mid$, and the functions $\mid\mid^a$ defined on synchronisation trees $\tau$ and $\tau' = \sum_j b_j.\tau_j$ by:

$$\tau \mid\mid^a \tau' = \sum_{b_j = a} a.(\tau \mid\mid \tau_j)$$

So we have a correct definition not in equational recursion format. So we must either

- find a different correct definition in the equational recursion format

or else

- find another algebraic format into which the correct definition fits.

When we come to the CSP parallel operator we do not even get as far as we did with synchronisation trees. The problem is like that with concealment: the distributive equation:

$$(F \,\Box\, F') \mid\mid G = (F \mid\mid G) \,\Box\, (F' \mid\mid G)$$

does not hold. One can show that there is no definition of $\mid\mid$ analogous to the above one for synchronisation trees, i.e., there is no suitable choice of algebra, $\Box_{\mathscr{A}}$ etc, and functions $\mid\mid^a$. The reason is that there is no binary operator $\Box'$ on (finitary) failure sets such that, for all $F,G,H$ we have:

$$(F \,\Box\, F') \,||\, G = (F \,||\, G) \,\Box'\, (F' \,||\, G)$$

For suppose, for the sake of contradiction, that there is such an operator. Then, fixing $F$ and $F'$, choose $G$ such that $F \,||\, G = F$, $F' \,||\, G = F'$ and $(F \,\Box\, F') \,||\, G = (F \,\Box\, F')$. Then, substituting into the above equation, we obtain that $F \,\Box\, F' = F \,\Box'\, F'$ and so the above equation yields distributivity, which, in fact, does not hold. As in the case of concealment, there may nonetheless be an indirect definition of $||$.

A similar difficulty obtains for the CSP interleaving operator. It too does not commute with $\Box$, and it too does not have any direct definition (the argument is like that for the concurrency operator but a little simpler, taking $G = \mathtt{Stop}$). As in the case of the concurrency operator, there may be an indirect definition.

## 5 Another choice of CSP effect constructors

Equations (3) and (4) do not immediately suggest a recursive definition of concealment. However, one can show that, for distinct actions $a_i$ ($i = 1,n$), the following equation holds between refusal sets:

$$(\Box_{i=1}^{n} a_i \to F_i)\backslash a_j = (F_j\backslash a_j) \sqcap ((F_j\backslash a_j) \,\Box\, \Box_{i\neq j} a_i \to (F_i\backslash a_j))$$

where $1 \leq j \leq n$. Taken together with equation (4), this suggests a recursive definition in terms of deterministic external choice. We therefore now change our choice of constructors, replacing binary external choice, action prefix and deadlock by deterministic external choice.

So as our second signature for CSP we take a binary operation symbol $\sqcap$ of internal choice and, for any *deterministic action sequence* $\vec{a}$ (i.e., any sequence of actions $a_i$ ($i = 1,n$), with the $a_i$ all different and $n \geq 0$), an $n$-ary operation symbol $\Box_{\vec{a}}$ of deterministic external choice. We write $\Box_{\vec{a}}(t_1,\ldots,t_n)$ as $\Box_{i=1}^{n} a_i t_i$ although it is more usual to use Hoare's notation $(a_1 \to t_1 \mid \cdots \mid a_n \to t_n)$; we also use $\mathtt{Stop}$ to abbreviate $\Box_{\vec{a}}()$.

We have the usual semilattice axioms for $\sqcap$. Deterministic external choice is commutative, in the sense that:

$$\Box_i a_i x_i = \Box_i a_{\pi(i)} x_{\pi(i)}$$

for any permutation $\pi$ of $\{1,\ldots,n\}$. Given this, we are justified in writing deterministic external choices over finite, possibly empty, sets of actions, $\Box_{a\in I} a t_a$, assuming some standard ordering of pairs $(a,t_a)$ without repetitions.

For the next axiom it is convenient to write $(a_1 \to t_1) \,\Box\, \Box_{i=2}^{n} a_i t_i$ for $\Box_{i=1}^{n} a_i t_i$ (for $n \geq 0$). The axiom states that deterministic external choice distributes over internal choice:

$$(a_1 \to (x \sqcap x')) \;\Box\; \Box_{i=2}^{n} a_i x_i = \left((a_1 \to x) \;\Box\; \Box_{i=2}^{n} a_i x_i\right) \;\sqcap\; \left((a_1 \to x') \;\Box\; \Box_{i=2}^{n} a_i x_i\right)$$

This implies that deterministic external choice is monotone with respect to $\sqsubseteq$.

We can regard a, possibly nondeterministic, external choice, in which the $a_i$ need not be all different, as an abbreviation for a deterministic one, via:

$$\Box_i a_i t_i = \Box_{b \in \{a_1,\ldots,a_n\}} b\left(\sqcap_{a_i = b} t_i\right) \tag{6}$$

With that convention we may also write $a_1 \to t_1 \;\Box\; \Box_{i=2}^{n} a_i t_i$ even when $a_1$ is some $a_i$, for $i > 1$. We can now write our final axiom:

$$\left(\Box_i a_i x_i\right) \;\sqcap\; \left((b_1 \to y_1) \Box \Box_{j=2}^{n} b_j y_j\right) \;\sqsubseteq\; (b_1 \to y_1) \;\Box\; \Box_i a_i x_i \tag{7}$$

Restricting the external choice $(b_1 \to y_1) \Box \Box_j b_j y_j$ to be deterministic gives an equivalent axiom, as does restricting $\Box_i a_i x_i$ (in the presence of the others).

Let us call this equational theory CSP($|$). The finitary refusal sets form a CSP($|$)-algebra $\mathscr{R}_{df}$ with the evident definitions:

$$\begin{aligned} F \sqcap_{\mathscr{R}_{df}} G &= F \cup G \\ (\Box_{\vec{a}})_{\mathscr{R}_{df}}(F_1,\ldots,F_n) &= \{(\varepsilon, W) \mid W \cap \{a_1,\ldots,a_n\} = \emptyset\} \cup \{(a_i w, W) \mid (w, W) \in F_i\} \end{aligned}$$

**Theorem 5.1.** *The finitary refusal sets algebra $\mathscr{R}_{df}$ is complete for equations between closed* CSP($|$) *terms.*

*Proof.* De Nicola's normal form can be regarded as written in the signature of CSP($|$), and a straightforward induction proves that every CSP($|$) term can be reduced to such a normal form using the above axioms. But two such normal forms have the same denotation whether they are regarded as CSP($\Box$) or as CSP($|$) terms, and in the former case, by Lemma 4.8 of [DeN85], they are identical. $\Box$

**Theorem 5.2.** *The finitary refusal sets algebra $\mathscr{R}_{df}$ is the initial* CSP($|$) *algebra.*

*Proof.* Following the proof of Lemma 3.2 we see that every finitary refusal set is definable by a closed CSP($|$) term. With that, initiality follows from the above completeness theorem, as in the proof of Theorem 3.4. $\Box$

Turning to the deconstructors, relabelling again has a straightforward homomorphic definition: given a relabelling function $f : A \to A$, $h_{Rl} : T_{\mathrm{CSP}(|)}(\emptyset) \to T_{\mathrm{CSP}(|)}(\emptyset)$ is defined homomorphically by:

$$h_{Rl}(F \sqcap G) = h_{Rl}(F) \sqcap h_{Rl}(G)$$

$$h_{Rl}(\bigsqcap_i a_i F_i) = \bigsqcap_i f(a_i) h_{Rl}(F_i)$$

As always one has to check that the implied algebra satisfies the equations, here those of CSP(|).

There is also now a natural homomorphic definition of concealment, $-\backslash a$, but, surprisingly perhaps, one needs to assume that $\Box$ is available. For every $a \in A$ one defines $h_a : T_{\mathrm{CSP}(|)}(\emptyset) \to T_{\mathrm{CSP}(|)}(\emptyset)$ homomorphically by:

$$h_a(F \sqcap G) = h_a(F) \sqcap h_a(G)$$

$$h_a\left(\bigsqcap_{i=1}^{n} a_i F_i\right) = \begin{cases} h_a(F_j) \sqcap (h_a(F_j) \Box \bigsqcap_{i \neq j} a_i h_a(F_i)) & (\text{if } a = a_j, \text{where } 1 \leq j \leq n) \\ \bigsqcap_{i=1}^{n} a_i h_a(F_i) & (\text{if } a \neq \text{any } a_i) \end{cases}$$

Verifying that the implicit algebra obeys satisfies the required equations is quite a bit of work. We record the result, but omit the calculations:

**Proposition 5.3.** *One can define a* CSP(|)*-algebra Con on* $T_{\mathrm{CSP}(|)}(\emptyset)$ *by:*

$$F \sqcap_{Con} G = F \sqcap G$$

$$(\bigsqcap_{\vec{a}})_{Con}(F_1, \ldots, F_n) = \begin{cases} F_j \sqcap (F_j \Box \bigsqcap_{i \neq j} a_i F_i) & (\textit{if } a = a_j) \\ \bigsqcap_i a_i F_i & (\textit{if } a \neq \textit{any } a_i) \end{cases}$$

The operator $\Box$ is, of course, no longer available as a constructor. However, it can alternatively be treated as a binary deconstructor. While its treatment as such is no more successful than our treatment of parallel operators, it is also no less successful. We define it simultaneously with $(n+1)$-ary functions $\Box^{a_1 \cdots a_n}$ on $T_{\mathrm{CSP}(|)}(\emptyset)$, for $n \geq 0$, where the $a_i$ are all distinct. That we are defining infinitely many functions simultaneously arises from dealing with the infinitely many deterministic choice operators (there would be be infinitely many even if we considered them as parameterised on the $a$'s). However, we anticipate that this will cause no real difficulty, given that we have overcome the difficulty of dealing with binary deconstructors.

Here are the required definitions:

$$(F \sqcap F') \Box G = (F \Box G) \sqcap (F' \Box G)$$
$$(\bigsqcap_i a_i F_i) \Box G = (F_1, \ldots, F_n) \Box^{a_1 \cdots a_n} G$$

$$(F_1, \ldots, F_n) \Box^{a_1 \cdots a_n} (G \sqcap G') = ((F_1, \ldots, F_n) \Box^{a_1 \cdots a_n} G) \sqcap ((F_1, \ldots, F_n) \Box^{a_1 \cdots a_n} G')$$
$$(F_1, \ldots, F_n) \Box^{a_1 \cdots a_n} (\bigsqcap_j b_j G_j) = (a_1 \to F_1) \Box (\ldots ((a_n \to F_n) \Box \bigsqcap_j b_j G_j) \ldots) \quad (8)$$

where, in the last equation, the notational convention $(a_1 \to t_1) \Box \bigsqcap_{i=2}^{n} a_i t_i$ is used $n$ times. It is clear that $\Box$ together with the functions

$$\Box^{a_1 \cdots a_n} : T_{\mathrm{CSP}(|)}(\emptyset)^{n+1} \to T_{\mathrm{CSP}(|)}(\emptyset)$$

defined by:

$$\Box^{a_1\ldots a_n}(F_1,\ldots,F_n,G) = (\Big\Box_i a_i F_i) \Box G \tag{9}$$

satisfy the equations, and, using the fact that all finitary refusal sets are definable by normal forms, one sees that they are the unique such functions.

We can treat the CSP parallel operator $||$ in a similar vein following the pattern given above for parallel merge operators in the case of synchronisation trees. We define it simultaneously with $(n+1)$-ary functions $||^{a_1\ldots a_n}$ on $T_{\mathrm{CSP}(||)}(\emptyset)$, for $n \geq 0$, where the $a_i$ are all distinct:

$$\begin{aligned}
(F \sqcap F') \,||\, G &= (F \,||\, G) \sqcap (F' \,||\, G)\\
(\Big\Box_i a_i F_i) \,||\, G &= (F_1,\ldots,F_n) \,||^{a_1\ldots a_n} G\\
(F_1,\ldots,F_n) \,||^{a_1\ldots a_n} (G \sqcap G') &= ((F_1,\ldots,F_n) \,||^{a_1\ldots a_n} G) \sqcap ((F_1,\ldots,F_n) \,\Box^{a_1\ldots a_n} G')\\
(F_1,\ldots,F_n) \,||^{a_1\ldots a_n} (\Big\Box_j b_j G_j) &= \Big\Box_{a_i=b_j} a_i(F_i \,||\, G_j)
\end{aligned} \tag{10}$$

Much as before, $||$ together with the functions $||^{a_1\ldots a_n}\colon T_{\mathrm{CSP}(||)}(\emptyset)^{n+1} \to T_{\mathrm{CSP}(||)}(\emptyset)$ defined by:

$$||^{a_1\ldots a_n} (F_1,\ldots,F_n,G) = (\Big\Box_i a_i F_i) \,||\, G$$

are the unique functions satisfying the equations.

Finally we consider the CSP interleaving operator $|||$. We define this by following an idea, exemplified in the ACP literature [BK85, BK86], of splitting an associative operation into several parts. Here we split $|||$ into a *left interleaving* operator $|||^l$ and a *right interleaving* operator $|||^r$ so that:

$$F \,|||\, G = (F \,|||^l\, G) \Box (F \,|||^r\, G)$$

In ACP the parallel operator is split into three parts: a left merge, a right merge (defined in terms of the left merge), and a communication merge; in a subtheory, PA, there is no communication, and the parallel operator, now an interleaving one, is split into left and right parts [BK86]. The idea of splitting an associative operation into several operations can be found in a much wider context [EFG08] where the split into two or three parts is axiomatised by the respective notions of dendriform dialgebra and trialgebra.

Our left and right interleaving are defined by the following 'binary deconstructor' equations:

$$
\begin{aligned}
(F \sqcap F') \,|||^l\, G &= (F \,|||^l\, G) \sqcap (F' \,|||^l\, G) \\
(\bigsqcap_{i=1}^{n} a_i F_i) \,|||^l\, G &= \bigsqcap_i a_i((F_i \,|||^l\, G) \,\Box\, (F_i \,|||^r\, G)) \\
G \,|||^r\, (F \sqcap F') &= (G \,|||^r\, F) \sqcap (G \,|||^r\, F') \\
G \,|||^r\, (\bigsqcap_{i=1}^{n} a_i F_i) &= \bigsqcap_i a_i((G \,|||^l\, F_i) \,\Box\, (G \,|||^r\, F_i)) \qquad (11)
\end{aligned}
$$

As may be expected, these equations also have unique solutions, now given by:

$$
\begin{aligned}
F \,|||^l\, G &= \{(\varepsilon, W) \mid (\varepsilon, W) \in F\} \cup \{(w, W) \mid (u, W) \in F,\ (v, W) \in G,\ w \in u \,|^l\, v\} \\
F \,|||^r\, G &= \{(\varepsilon, W) \mid (\varepsilon, W) \in G\} \cup \{(w, W) \mid (u, W) \in F,\ (v, W) \in G,\ w \in u \,|^r\, v\}
\end{aligned}
$$

where $u \,|^l\, v$ is the set of interleavings of $u$ and $v$ which begin with a letter of $u$, and $u \,|^r\, v$ is defined analogously. It is interesting to note that:

$$F \,|||^l\, (G \sqcap G') = (F \,|||^l\, G) \sqcap (F \,|||^l\, G')$$

and similarly for $|||^r$.

## 6 Adding divergence

The treatment of CSP presented thus far dealt with finite divergence-free processes only. There are several ways to extend the refusal sets model of Section 3 to infinite processes with divergence. The most well-known model is the *failures/divergences* model of [Hoa85], further elaborated in [Ros98]. A characteristic property of this model is that divergence, i.e., an infinite sequence of internal actions, is modelled as *Chaos*, a process that satisfies the equation:

$$Chaos \,\Box\, x = Chaos \sqcap x = Chaos \qquad (12)$$

So after *Chaos* no further process activity is discernible.

An alternative extension is the *stable failures* model proposed in [BKO87], and also elaborated in [Ros98]. This model equates processes that allow the same *observations*, where actions and deadlock are considered observable, but divergence does not give rise to any observations. A failure pair $(w, W)$—now allowing $W$ to be infinite—records an observation in which $w$ represents a sequence of actions being observed, and $W$ represents the observation of deadlock under the assumption that the environment in which the observed process is running allows only the (inter)actions in the set $W$. Such an observation can be made if after engaging in the sequence of visible actions $w$, the observed process reaches a state in which no further internal actions are possible, nor any actions from the set $W$. Besides failure pairs, also traces are observable, and thus the observable behaviour of a process is

given by a pair $(T,F)$ where $T$ is a set of traces and $F$ is a set of failure pairs. Unlike the model $\mathscr{R}_f$ of Section 3, the traces are not determined by the failure pairs. In fact, in a process that can diverge in every state, the set of failure pairs is empty, yet the set of traces conveys important information.

In the remainder of this paper we add a constant $\Omega$ to the signature of CSP that is a zero for the semilattice generated by $\sqcap$. This will greatly facilitate the forthcoming development. Intuitively, one may think of $\Omega$ as divergence in the stable failures model.

W.r.t. the equational theory CSP$(\Box)$ of Section 3 we thus add the constant $\Omega$ and the single axiom:

$$x \sqcap \Omega = x \tag{13}$$

thereby obtaining the theory CSP$(\Box,\Omega)$. We note two useful derived equations:

$$\begin{aligned} x \sqcap (\Omega \Box y) &= x \sqcap (x \Box y) \\ (\Omega \Box x) \sqcap (\Omega \Box y) &= (\Omega \Box x) \Box (\Omega \Box y) \end{aligned} \tag{14}$$

Semantically, a *process* is now given by a pair $(T,F)$, where $T$ is a set of traces and $F$ is a set of failure pairs that satisfy the following conditions:

1. $\varepsilon \in T$
2. $wa \in T \Rightarrow w \in T$
3. $(w,W) \in F \Rightarrow w \in T$
4. $(w,W) \in F \wedge V \subseteq W \Rightarrow (w,V) \in F$
5. $(w,W) \in F \wedge \forall a \in V.\, wa \notin T \Rightarrow (w, W \cup V) \in F$ (where $V \subseteq A$)

The two components of such a pair $P$ are denoted $T_P$ and $F_P$, respectively, and for $w \in T_P$ we define $\mathrm{fut}_P(w) := \{a \in A \mid wa \in T_P\}$. We can define the CSP operators on processes by setting

$$P \,\mathrm{op}\, Q = (P \,\mathrm{op}_{\mathscr{T}}\, Q, P \,\mathrm{op}_{\mathscr{R}}\, Q)$$

where $\mathrm{op}_{\mathscr{T}}$ is given by:

$$\begin{aligned} \mathtt{Stop}_{\mathscr{T}} &= \{\varepsilon\} \\ a \rightarrow_{\mathscr{T}} P &= \{\varepsilon\} \cup \{aw \mid w \in T_P\} \\ P \sqcap_{\mathscr{T}} Q &= T_P \cup T_Q \\ P \Box_{\mathscr{T}} Q &= T_P \cup T_Q \\ f_{\mathscr{T}}(P) &= \{f(w) \mid w \in T_P\} \\ P \backslash_{\mathscr{T}} a &= \{w \backslash a \mid w \in T_P\} \\ P \,||_{\mathscr{T}}\, Q &= \{w \mid w \in T_P,\ w \in T_Q\} \\ P \,|||_{\mathscr{T}}\, Q &= \{w \mid u \in T_P,\ v \in T_Q,\ w \in u \,|\, v\} \end{aligned}$$

and $\mathrm{op}_{\mathscr{R}}$ is given as $\mathrm{op}_{\mathscr{R}_f}$ was in Section 3, but without the restriction to finite sets $W$ in defining $\mathtt{Stop}_{\mathscr{R}}$. For the new process $\Omega$ we set

$$\Omega_{\mathscr{T}} = \{\varepsilon\} \qquad \text{and} \qquad \Omega_{\mathscr{R}} = \emptyset$$

This also makes the collection of processes into a CSP$(\Box, \Omega)$-algebra, $\mathscr{F}$.

A process $P$ is called *finitary* if $T_P$ is finite. The finitary processes evidently form a subalgebra of $\mathscr{F}$; we call it $\mathscr{F}_f$.

**Lemma 6.1.** *Let $P$ be a finitary process. Then, for every $w \in T_P$ there is an $n \geq 0$ and $V_1, \ldots, V_n \subseteq \mathrm{fut}_F(w)$ such that $(w, W) \in F_P$ iff $W \cap V_i = \emptyset$ for some $i \in \{1, \ldots, n\}$.*

*Proof.* Closure conditions 4 and 5 above imply that $(w, W) \in F_P$ if, and only if, $(w, W \cap \mathrm{fut}_P(w)) \in F_P$. Thus we only need to be concerned about pairs $(w, W)$ with $W \subseteq \mathrm{fut}_P(w)$. Now, as $\mathrm{fut}_P(w)$ is finite, for any relevant $(w, W) \in F$, of which there are finitely many, we can take $V$ to be $\mathrm{fut}_P(w) \backslash W$, and we obtain finitely many such sets. □

Note that it may happen that $n = 0$, in contrast with the case of Lemma 3.1.

**Lemma 6.2.** *All finitary processes are definable by closed* CSP$(\Box, \Omega)$ *terms.*

*Proof.* Let $P$ be a finitary process. We proceed by induction on the length of the longest trace in $T_P$. By the previous lemma there are sets $V_1, \ldots, V_n$, for some $n \geq 0$, such that $(\varepsilon, W) \in F$ iff $W \cap V_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. Define $T_a$ and $F_a$, for $a \in T_P$, by:

$$T_a = \{w \mid aw \in T_P\} \qquad F_a = \{(w, W) \mid (aw, W) \in F_P\}$$

Then it is not hard to see that each $P_a := (T_a, F_a)$ is a finitary process, and that

$$P = \left( \bigsqcap_i \Box_{a \in V_i} a \to P_a \right) \sqcap \left( \Omega \Box \Box_{a \in T_P} a \to P_a \right)$$

As the longest trace in $T_a$ is strictly shorter than the longest one in $T_P$, the proof concludes, employing the induction hypothesis. □

**Proposition 6.3.** CSP$(\Box, \Omega)$ *is ground equationally complete with respect to both $\mathscr{F}$ and $\mathscr{F}_f$.*

*Proof.* This time we recursively define a normal form as a CSP$(\Box, \Omega)$-term of the form

$$\bigsqcap_{L \in \mathscr{L}} \Box_{a \in L} a \to t_a \qquad \text{or} \qquad \Omega \Box \Box_{a \in K} a \to t_a$$

where $\mathscr{L}$ is a finite non-empty saturated collection of finite sets of actions, $K$ is a finite set of actions, and each term $t_a$ is in normal form. Every term is provably equal in CSP$(\Box, \Omega)$ to a term in normal form; the proof proceeds as for Proposition 3.3, but now also using the derived equations (14). Next, by Lemma 7.2 below, if two normal forms have the same denotation in $\mathscr{F}$ then they are identical. So the result follows for $\mathscr{F}$, and then for $\mathscr{F}_f$ too, as all closed terms denote finitary processes. □

**Theorem 6.4.** *The algebra $\mathscr{F}_f$ of finitary processes is the initial* $\mathrm{CSP}(\Box,\Omega)$ *algebra.*

*Proof.* Let the initial such algebra be I. There is a unique homomorphism $h\colon \mathrm{I} \to \mathscr{F}_f$. By Lemma 6.2, $h$ is a surjection. By the previous proposition, $\mathscr{F}_f$ is complete for equations between closed terms, and so $h$ is an injection. Hence $h$ is an isomorphism, completing the proof. □

As in Section 5, in order to deal with deconstructors, particularly hiding, we replace external choice by deterministic external choice. The availability of $\Omega$ permits useful additional such operators. The equational theory $\mathrm{CSP}(|,\Omega)$ has as signature the binary operation symbol $\sqcap$, and for any deterministic action sequence $\vec{a}$, the $n$-ary operation symbols $\Box_{\vec{a}}$ (as in Section 5), as well as the new $n$-ary operation symbols $\Box^{\Omega}_{\vec{a}}$, for $n \geq 0$, which denote a deterministic external choice with $\Omega$ as one of the summands. We adopt conventions for $\Box^{\Omega}_{\vec{a}}$ analogous to those previously introduced for $\Box_{\vec{a}}(t_1,\ldots,t_n)$. We write $\Box^{\Omega}_{\vec{a}}(t_1,\ldots,t_n)$ as $\Omega \Box \Box_{i=1}^{n} a_i t_i$. We also write $\Omega \Box (c_1 \to t_1) \Box \Box_{j=2}^{n} c_j t_j$ for $\Omega \Box \Box_{j=1}^{n} c_j t_j$, so that the $c_j$ $(j = 1,n)$ must all be distinct.

The first three groups of axioms of $\mathrm{CSP}(|,\Omega)$ are:

- $\sqcap, \Omega$ is a semilattice with a zero—here $\Omega$ is the 0-ary case of $\Box^{\Omega}_{\vec{a}}$,
- both deterministic external choice operators $\Box_{\vec{a}}$ and $\Box^{\Omega}_{\vec{a}}$ are commutative, as explained in Section 5, and
- both deterministic external choice operators distribute over internal choice, as explained in Section 5,

Given commutativity, we are, as before, justified in writing deterministic external choices $\Box_{a\in I} a t_a$ or $\Omega \Box \Box_{a\in I} a t_a$, over finite, possibly empty, sets of actions $I$, assuming some standard ordering of pairs $(a,t_a)$ without repetitions. Next, using the analogous convention to (6) we can then also understand $\Omega \Box \Box_{j=1}^{n} c_j t_j$, and so also $\Omega \Box (c_1 \to t_1) \Box \Box_{j=2}^{n} c_j t_j$, even when the $c_j$ are not all distinct. With these conventions established, we can now state the final group of axioms. These are all variants of Axiom (7) of Section 5, allowing each of the two deterministic external choices to have an $\Omega$-summand:

$$\left(\Omega \Box \Box_i a_i x_i\right) \sqcap \left(\Omega \Box (b_1 \to y_1) \Box \Box_{j=2}^{n} b_j y_j\right) \sqsubseteq \Omega \Box (b_1 \to y_1) \Box \Box_i a_i x_i$$

$$\left(\Omega \Box \Box_i a_i x_i\right) \sqcap \left((b_1 \to y_1) \Box \Box_{j=2}^{n} b_j y_j\right) \sqsubseteq \Omega \Box (b_1 \to y_1) \Box \Box_i a_i x_i$$

$$\left(\Box_i a_i x_i\right) \sqcap \left(\Omega \Box (b_1 \to y_1) \Box \Box_{j=2}^{n} b_j y_j\right) \sqsubseteq (b_1 \to y_1) \Box \Box_i a_i x_i$$

$$\left(\bigsqcap_i a_i x_i\right) \sqcap \left((b_1 \to y_1) \sqcap \bigsqcap_{j=2}^{n} b_j y_j\right) \sqsubseteq (b_1 \to y_1) \sqcap \bigsqcap_i a_i x_i \tag{15}$$

As in the case of Axiom (7), restricting any of these choices to be deterministic results in an axiom of equivalent power. We note two useful derived equations:

$$\begin{aligned} \bigsqcap_i a_i x_i \sqcap (\Omega \sqcap \bigsqcap_j b_j y_j) &= \bigsqcap_i a_i x_i \sqcap (\bigsqcap_i a_i x_i \sqcap \bigsqcap_j b_j y_j) \\ (\Omega \sqcap \bigsqcap_i a_i x_i) \sqcap (\Omega \sqcap \bigsqcap_j b_j y_j) &= (\Omega \sqcap \bigsqcap_i a_i x_i) \sqcap \bigsqcap_j b_j y_j \end{aligned} \tag{16}$$

where two further notational conventions are employed: $(\bigsqcap_{i=1}^{m} a_i t_i) \sqcap (\bigsqcap_{j=1}^{n} b_j t'_j)$ stands for $\bigsqcap_{k=1}^{m+n} c_k t''_k$ where $c_k = a_k$ and $t''_k = t_k$, for $k = 1, m$, and $c_k = b_{k-m}$, and $t''_k = t'_{k-m}$, for $k = m+1, m+n$; and $(\Omega \sqcap \bigsqcap_{i=1}^{m} a_i t_i) \sqcap (\bigsqcap_{j=1}^{n} b_j t'_j)$ is understood analogously. In fact, the first three axioms of (15) are also derivable from (16), in the presence of the other axioms, and thus may be replaced by (16).

The collection of processes is turned into a $\mathrm{CSP}(|,\Omega)$-algebra $\mathscr{F}_d$ as before, writing:

$$P \operatorname{op}_{\mathscr{F}_d} Q = (P \operatorname{op}_{\mathscr{T}_d} Q, P \operatorname{op}_{\mathscr{R}_d} Q)$$

and defining $\operatorname{op}_{\mathscr{T}_d}$ and $\operatorname{op}_{\mathscr{R}_d}$ in the evident way:

$$\begin{aligned} P \sqcap_{\mathscr{T}_d} Q &= T_P \cup T_Q \\ (\bigsqcap_{\vec{a}})_{\mathscr{T}_d}(P_1, \ldots, P_n) &= \{\varepsilon\} \cup \{a_i w \mid w \in T_{P_i}\} \\ (\bigsqcap_{\vec{a}}^{\Omega})_{\mathscr{T}_d}(P_1, \ldots, P_n) &= \{\varepsilon\} \cup \{a_i w \mid w \in T_{P_i}\} \\ (\bigsqcap_{\vec{a}}^{\Omega})_{\mathscr{R}_d}(P_1, \ldots, P_n) &= \{(a_i w, W) \mid (w, W) \in F_{P_i}\} \end{aligned}$$

with $\sqcap_{\mathscr{R}_d}$ and $(\bigsqcap_{\vec{a}})_{\mathscr{R}_d}$ given just as in Section 5. Exactly as in Section 5, but now using the derived equations (16), we obtain:

**Theorem 6.5.** *The algebra $\mathscr{F}_d$ is complete for equations between closed* $\mathrm{CSP}(|,\Omega)$ *terms.*

**Theorem 6.6.** *The finitary subalgebra $\mathscr{F}_{df}$ of $\mathscr{F}_d$ is the initial* $\mathrm{CSP}(|,\Omega)$ *algebra.*

Turning to the deconstructors, relabelling and concealment can again be treated homomorphically. For relabelling by $f$ one simply adds the equation:

$$h_{Rl}(\Omega \sqcap \bigsqcap_i a_i F_i) = \Omega \sqcap \bigsqcap_i f(a_i) h_{Rl}(F_i)$$

to the treatment in Section 5, and checks that the implied algebra satisfies the equations. Pleasingly, the treatment of concealment can be simplified in such a way that the deconstructor $\sqcap$ is no longer needed. For every $a \in A$ one defines $h_a : T_{\mathrm{CSP}(|,\Omega)}(\emptyset) \to T_{\mathrm{CSP}(|,\Omega)}(\emptyset)$ homomorphically by:

$$h_a(P \sqcap Q) = h_a(P) \sqcap h_a(Q)$$

$$h_a\left(\sum_{i=1}^{n} a_i P_i\right) = \begin{cases} h_a(P_j) \sqcap (\Omega \Box \sum_{i \neq j} a_i h_a(P_i)) & (\text{if } a = a_j, \text{where } 1 \leq j \leq n) \\ \sum_{i=1}^{n} a_i h_a(P_i) & (\text{if } a \neq \text{any } a_i) \end{cases}$$

$$h_a\left(\Omega \Box \sum_{i=1}^{n} a_i P_i\right) = \begin{cases} h_a(P_j) \sqcap (\Omega \Box \sum_{i \neq j} a_i h_a(P_i)) & (\text{if } a = a_j, \text{where } 1 \leq j \leq n) \\ \Omega \Box \sum_{i=1}^{n} a_i h_a(P_i) & (\text{if } a \neq \text{any } a_i) \end{cases}$$

Note the use of the new form of deterministic choice here. One has again to verify that the implicit algebra obeys satisfies the required equations. The treatment of the binary deconstructors $\Box$, $||$ and $|||$ is also a trivial adaptation of the treatment in Section 5. For $\Box$ one adds a further auxiliary operator $\Box^{\Omega,a_1\ldots a_n}$ and the equations:

$$(\Omega \Box \sum_i a_i P_i) \Box Q = (P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} Q$$

$$(P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} (Q \sqcap Q') = ((P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} Q) \sqcap ((P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} Q')$$

$$(P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} (\sum_j b_j Q_j) = (\Omega \Box \sum_i a_i P_i) \Box \sum_j b_j Q_j$$

$$(P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} (\Omega \Box \sum_j b_j Q_j) = (\Omega \Box \sum_i a_i P_i) \Box \sum_j b_j Q_j$$

$$(P_1, \ldots, P_n) \Box^{a_1\ldots a_n} (\Omega \Box \sum_j b_j Q_j) = (\Omega \Box \sum_i a_i P_i) \Box \sum_j b_j Q_j$$

For $||$ one adds the auxiliary operator $||^{\Omega,a_1\ldots a_n}$ and the equations:

$$(\Omega \Box \sum_i a_i P_i) \,||\, Q = (P_1, \ldots, P_n) \,||^{\Omega,a_1\ldots a_n} Q$$

$$(P_1, \ldots, P_n) \,||^{\Omega,a_1\ldots a_n} (Q \sqcap Q') = ((P_1, \ldots, P_n) \,||^{\Omega,a_1\ldots a_n} Q) \sqcap ((P_1, \ldots, P_n) \Box^{\Omega,a_1\ldots a_n} Q')$$

$$(P_1, \ldots, P_n) \,||^{\Omega,a_1\ldots a_n} (\sum_j b_j Q_j) = \Omega \Box \sum_{a_i = b_j} a_i (P_i \,||\, Q_j)$$

$$(P_1, \ldots, P_n) \,||^{\Omega,a_1\ldots a_n} (\Omega \Box \sum_j b_j Q_j) = \Omega \Box \sum_{a_i = b_j} a_i (P_i \,||\, Q_j)$$

$$(P_1, \ldots, P_n) \,||^{a_1\ldots a_n} (\Omega \Box \sum_j b_j Q_j) = \Omega \Box \sum_{a_i = b_j} a_i (P_i \,||\, Q_j)$$

Finally, for $|||$ one simply adds extra equations:

$$(\Omega \,\square \bigsqcap_{i=1}^{n} a_i P_i) \,|||^l\, Q = \Omega \,\square \bigsqcap_i a_i((P_i \,|||^l\, Q) \,\square\, (P_i \,|||^r\, Q))$$

$$Q \,|||^r\, (\Omega \,\square \bigsqcap_{i=1}^{n} a_i P_i) = \Omega \,\square \bigsqcap_i a_i((Q \,|||^l\, P_i) \,\square\, (Q \,|||^r\, P_i))$$

## 7 Combining CSP and functional programming

To combine CSP with functional programming, specifically the computational $\lambda$-calculus, we use the monad $T_{\mathrm{CSP}(|,\Omega)}$ for the denotational semantics. As remarked above, CSP processes then become terms of type `empty`. However, as the constructors are polymorphic, it is natural to go further and look for polymorphic versions of the deconstructors. We therefore add polymorphic constructs to $\lambda_c$ as follows:

**Constructors**

$$\frac{M:\sigma \quad N:\sigma}{M \sqcap N:\sigma} \qquad \frac{M:\sigma}{a \to M:\sigma} \qquad \Omega:\sigma$$

**Unary Deconstructors**

$$\frac{M:\sigma}{f(M):\sigma} \qquad \frac{M:\sigma}{M\backslash a:\sigma}$$

for any relabelling function $f$, and any $a \in A$. (One should really restrict the allowable relabelling functions in order to keep the syntax finitary.)

**Binary Deconstructors**

$$\frac{M:\sigma \quad N:\sigma}{M \,\square\, N:\sigma} \qquad \frac{M:\sigma \quad N:\tau}{M \,||\, N:\sigma\times\tau} \qquad \frac{M:\sigma \quad N:\tau}{M \,|||\, N:\sigma\times\tau}$$

The idea of the two parallel constructs is to evaluate the two terms in parallel and then return the pair of the two values produced. We did not include syntax for the two deterministic choice constructors as they are definable from $a \to -$ and $\Omega$ with the aid of the $\square$ deconstructor.

For the denotational semantics, the semantics of types is given as usual using the monad $T_{\mathrm{CSP}(|,\Omega)}$, which we know exists by the general considerations of Section 2. These general considerations also yield a semantics for the constructors. For example, for every set $X$ we have the map:

$$\sqcap_X : T_{\mathrm{CSP}(|,\Omega)}(X)^2 \to T_{\mathrm{CSP}(|,\Omega)}(X)$$

which we can use for $X = [\![\sigma]\!]$ to interpret terms $M \sqcap N : \sigma$.

The homomorphic point of view also leads to an interpretation of the unary deconstructors, but using free algebras rather than just the initial one. For example, for

relabelling by $f$ we need a function:

$$h_{Rl} : T_{\mathrm{CSP}(|,\Omega)}(X) \to T_{\mathrm{CSP}(|,\Omega)}(X)$$

We obtain this as the unique homomorphism extending the unit $\eta_X : X \to T_{\mathrm{CSP}(|,\Omega)}(X)$, equipping $T_{\mathrm{CSP}(|,\Omega)}(X)$ with the algebra structure $\mathscr{A} = (T_{\mathrm{CSP}(|,\Omega)}(X), \sqcap_{\mathscr{A}}, \Box_{\mathscr{A}}, \Box^{\Omega}_{\mathscr{A}})$ where

$$x \sqcap_{\mathscr{A}} y = x \sqcap_X y$$

for $x, y \in T_{\mathrm{CSP}(|,\Omega)}(X)$,

$$(\Box_{\vec{a}})_{\mathscr{A}}(x_1, \ldots, x_n) = (\Box_{f(\vec{a})})_X(x_1, \ldots, x_n)$$

and

$$(\Box^{\Omega}_{\vec{a}})_{\mathscr{A}}(x_1, \ldots, x_n) = (\Box^{\Omega}_{f(\vec{a})})_X(x_1, \ldots, x_n)$$

Concealment $-\backslash a$ can be treated analogously, but now following the treatment in the case of $\mathscr{F}_{df}$, and defining $\mathscr{A}$ by:

$$x \sqcap_{\mathscr{A}} y = x \sqcap_X y$$

for $x, y \in T_{\mathrm{CSP}(|,\Omega)}(X)$,

$$(\Box_{\vec{a}})_{\mathscr{A}}(x_1, \ldots, x_n) = \begin{cases} x_j \sqcap (\Omega \Box \Box_{i \neq j} a_i x_i) & (\text{if } a = a_j, \text{ where } 1 \leq j \leq n) \\ \Box_{i=1}^{n} a_i x_i & (\text{if } a \neq \text{any } a_i) \end{cases}$$

and

$$(\Box^{\Omega}_{\vec{a}})_{\mathscr{A}}(x_1, \ldots, x_n) = \begin{cases} x_j \sqcap (\Omega \Box \Box_{i \neq j} a_i x_i) & (\text{if } a = a_j, \text{ where } 1 \leq j \leq n) \\ \Omega \Box \Box_{i=1}^{n} a_i x_i & (\text{if } a \neq \text{any } a_i) \end{cases}$$

We here again make use of the deterministic choice operator made available by the presence of $\Omega$.

However, we cannot, of course, carry this on to binary deconstructors as we have no general algebraic treatment of them. We proceed instead by giving a concrete definition of them (and the other constructors and deconstructors). That is, we give an explicit description of the free $\mathrm{CSP}(|,\Omega)$-algebra on a set $X$ and define our operators in terms of that representation.

An *$X$-trace* is a pair $(w,x)$, where $w \in A^*$ and $x \in X$; it is generally more suggestive to write $(w,x)$ as $wx$. For any relabelling function $f$, we set $f(wx) = f(w)x$, and, for any $a \in A$, we set $wx\backslash a = (w\backslash a)x$. An *$X$-process* is a pair $(T,F)$ with $T$ a set of traces as well as $X$-traces, and $F$ a set of failure pairs, satisfying the same five conditions as in Section 6, together with:

$2'$ $wx \in T \Rightarrow w \in T$ (for $x \in X$)

The CSP operators are defined on $X$-processes exactly as before, except that the two parallel operators now have more general types:

$$||_{X,Y}, |||_{X,Y} : T_{\mathrm{CSP}(|,\Omega)}(X) \times T_{\mathrm{CSP}(|,\Omega)}(Y) \to T_{\mathrm{CSP}(|,\Omega)}(X \times Y)$$

We take $\mathrm{fut}_P(w) := \{a \in A \mid wa \in T_P\}$, as before.

$$
\begin{array}{lcl}
\Omega_{\mathscr{T}(X)} & = & \{\varepsilon\} \\
\Omega_{\mathscr{R}(X)} & = & \emptyset \\
\mathtt{Stop}_{\mathscr{T}(X)} & = & \{\varepsilon\} \\
\mathtt{Stop}_{\mathscr{R}(X)} & = & \{(\varepsilon, W) \mid W \subseteq A\} \\
a \to_{\mathscr{T}(X)} P & = & \{\varepsilon\} \cup \{aw \mid w \in T_P\} \\
a \to_{\mathscr{R}(X)} P & = & \{(\varepsilon, W) \mid a \notin W\} \cup \{(aw, W) \mid (w, W) \in F_P\} \\
P \sqcap_{\mathscr{T}(X)} Q & = & T_P \cup T_Q \\
P \sqcap_{\mathscr{R}(X)} Q & = & F_P \cup F_Q \\
P \,\Box_{\mathscr{T}(X)}\, Q & = & T_P \cup T_Q \\
P \,\Box_{\mathscr{R}(X)}\, Q & = & \{(\varepsilon, W) \mid (\varepsilon, W) \in F_P \cap F_Q\} \cup \{(w, W) \mid w \neq \varepsilon,\ (w, W) \in F_P \cup F_Q\} \\
f_{\mathscr{T}(X)}(P) & = & \{f(w) \mid w \in T_P\} \\
f_{\mathscr{R}(X)}(P) & = & \{(f(w), W) \mid (w, f^{-1}(W) \cap \mathrm{fut}_P(w)) \in F_P\} \\
P \backslash_{\mathscr{T}(X)} a & = & \{w \backslash a \mid w \in T_P\} \\
P \backslash_{\mathscr{R}(X)} a & = & \{(w \backslash a, W) \mid (w, W \cup \{a\}) \in F_P\} \\
P \,||_{\mathscr{T}(X,Y)}\, Q & = & \{w \mid w \in T_P \cap T_Q \cap A^*\} \cup \{w(x,y) \mid wx \in T_P,\, wy \in T_Q\} \\
P \,||_{\mathscr{R}(X,Y)}\, Q & = & \{(w, W \cup V) \mid (w, W) \in F_P,\ (w, V) \in F_Q\} \\
P \,|||_{\mathscr{T}(X,Y)}\, Q & = & \{w \mid u \in T_P \cap A^*,\ v \in T_Q \cap A^*,\ w \in u \,|\, v\} \cup \\
 & & \qquad \{w(x,y) \mid ux \in T_P,\ vy \in T_Q,\ w \in u \,|\, v\} \\
P \,|||_{\mathscr{R}(X,Y)}\, Q & = & \{(w, W) \mid (u, W) \in F_P, (v, W) \in F_Q, w \in u \,|\, v\}
\end{array}
$$

Here, much as before, we write $P \,\mathrm{op}_{\mathscr{F}(X)}\, Q = (P \,\mathrm{op}_{\mathscr{T}(X)}\, Q, P \,\mathrm{op}_{\mathscr{R}(X)}\, Q)$ when defining the CSP operators on $X$-processes. The $X$-processes also form the carrier of a $\mathrm{CSP}(|, \Omega)$-algebra $\mathscr{F}_d(X)$, with the operators defined as follows:

$$
\begin{array}{lcl}
P \sqcap_{\mathscr{T}_d(X)} Q & = & T_P \cup T_Q \\
P \sqcap_{\mathscr{R}_d(X)} Q & = & F_P \cup F_Q \\
(\Box_{\vec{a}}^{\Omega})_{\mathscr{T}_d(X)}(P_1, \ldots, P_n) & = & \{\varepsilon\} \cup \{a_i w \mid w \in T_{P_i}\} \\
(\Box_{\vec{a}}^{\Omega})_{\mathscr{R}_d(X)}(P_1, \ldots, P_n) & = & \{(a_i w, W) \mid (w, W) \in F_{P_i}\} \\
(\Box_{\vec{a}})_{\mathscr{T}_d(X)}(P_1, \ldots, P_n) & = & \{\varepsilon\} \cup \{a_i w \mid w \in T_{P_i}\} \\
(\Box_{\vec{a}})_{\mathscr{R}_d(X)}(P_1, \ldots, P_n) & = & \{(\varepsilon, W) \mid W \cap \{a_1, \ldots, a_n\} = \emptyset\} \cup \\
 & & \qquad \{(a_i w, W) \mid (w, W) \in F_{P_i}\}
\end{array}
$$

The finitary $X$-processes are those with a finite set of traces and $X$-traces; they form the carrier of a $\mathrm{CSP}(|, \Omega)$-algebra $\mathscr{F}_{df}(X)$.

We now show that $\mathscr{F}_{df}(X)$ is the free $\mathrm{CSP}(|, \Omega)$-algebra over $X$. As is well known, the free algebra of a theory Th over a set $X$ is the same as the initial algebra of the theory $\mathrm{Th}^+$ obtained by extending Th with constants $\underline{x}$ for each $x \in X$ but without changing the axioms. The unit map $\eta : X \to T_{\mathrm{Th}}(X)$ sends $x \in X$ to the denotation of $\underline{x}$ in the initial algebra. We therefore show that $\mathscr{F}_{df}(X)$, extended to a $\mathrm{CSP}(|, \Omega)^+$-algebra by taking

$$[\![\underline{x}]\!] = (\{x\}, \emptyset) \qquad (\text{for } x \in X)$$

is the initial $\mathrm{CSP}(|,\Omega)^+$-algebra. We begin by looking at definability.

**Lemma 7.1.** *The finitary $X$-processes are those definable by closed* $\mathrm{CSP}(|,\Omega)^+$ *terms.*

*Proof.* The proof goes just as the one for Lemma 6.2, using that Lemma 6.1 applies just as well to finitary $X$-processes, but this time we have

$$P = \bigsqcap_i \bigsqcap_{a\in V_i} a \to P_a \sqcap \left( \Omega \Box \bigsqcap_{a\in T_P} a \to P_a \right) \sqcap \bigsqcap_{x\in T_P} \underline{x} \qquad \Box$$

Next, we say that a closed $\mathrm{CSP}(|,\Omega)^+$-term $t$ is in *normal form* if it is has one of the following two forms:

$$\bigsqcap_{L\in\mathscr{L}} \bigsqcap_{a\in L} at_a \sqcap \bigsqcap_{x\in J} \underline{x} \qquad \text{or} \qquad \left( \Omega \Box \bigsqcap_{a\in K} at_a \right) \sqcap \bigsqcap_{x\in J} \underline{x}$$

where, as appropriate, $\mathscr{L}$ is a finite non-empty saturated collection of finite sets of actions, $J \subseteq_{\mathrm{fin}} X$, $K \subseteq_{\mathrm{fin}} A$, and each term $t_a$ is in normal form.

**Lemma 7.2.** *Two normal forms are identical if they have the same denotation in* $\mathscr{F}_{df}(X)$.

*Proof.* Consider two normal forms with the same denotation in $\mathscr{F}_{df}(X)$, say $(T,F)$. As $(\varepsilon,\emptyset) \in F$ iff $F$ is the denotation of a normal form of the first form (rather than the second), both normal forms must be of the same form. Thus, there are two cases to consider, the first of which concerns two forms:

$$\bigsqcap_{L\in\mathscr{L}} \bigsqcap_{a\in L} at_a \sqcap \bigsqcap_{x\in J} \underline{x} \qquad\qquad \bigsqcap_{L'\in\mathscr{L}'} \bigsqcap_{a'\in L'} a't'_{a'} \sqcap \bigsqcap_{x\in J'} \underline{x}$$

We argue by induction on the sum of the sizes of the two normal forms. We evidently have that $J = J'$. Next, if $a \in \bigcup\mathscr{L}$ then $a \in T$ and so $a \in \bigcup\mathscr{L}'$; we therefore have that $\bigcup\mathscr{L} \subseteq \bigcup\mathscr{L}'$. Now, if $L \in \mathscr{L}$ then $(\varepsilon, (\bigcup\mathscr{L}')\backslash L) \in F$; so for some $L' \in \mathscr{L}$ we have $L' \cap ((\bigcup\mathscr{L}')\backslash L) = \emptyset$, and so $L' \subseteq L$. As $\mathscr{L}'$ is saturated, it follows by the previous remark that $L \in \mathscr{L}'$. So we have the inclusion $\mathscr{L} \subseteq \mathscr{L}'$ and then, arguing symmetrically, equality.

Finally, the denotations of $t_a$ and $t'_a$, for $a \in \bigcup\mathscr{L} = \bigcup\mathscr{L}'$ are the same, as they are determined by $T$ and $F$, being $\{w \mid aw \in T\}$ and $\{(w,W) \mid (aw,W) \in F\}$, and the argument concludes, using the inductive hypothesis.

The other case concerns normal forms:

$$\left( \Omega \Box \bigsqcap_{a\in K} at_a \right) \sqcap \bigsqcap_{x\in J} \underline{x} \qquad\qquad \left( \Omega \Box \bigsqcap_{a'\in K'} a't'_a \right) \sqcap \bigsqcap_{x\in J'} \underline{x}$$

Much as before we find $J = J'$, $K = K'$, and $t_a = t_a$ for $a \in K$. $\Box$

**Lemma 7.3.** $\mathrm{CSP}(|,\Omega)^+$ *is ground complete with respect to* $\mathscr{F}_{df}(X)$.

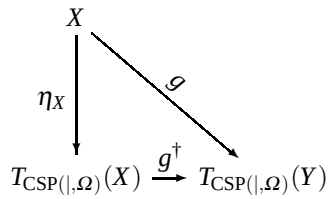
*Proof.* As before, a straightforward induction shows that every term has a normal form, and then completeness follows by Lemma 7.2. □

**Theorem 7.4.** *The algebra* $\mathscr{F}_{df}(X)$ *is the free* $\mathrm{CSP}(|,\Omega)$*-algebra over* $X$.

*Proof.* It follows from Lemmas 7.1 and 7.3 that $\mathscr{F}_{df}(X)^+$ is the initial $\mathrm{CSP}(|,\Omega)^+$-algebra. □

As with any finitary equational theory, $\mathrm{CSP}(|,\Omega)$ is equationally complete with respect to $\mathscr{F}_{df}(X)$ when $X$ is infinite. It is not difficult to go a little further and show that this also holds when $X$ is only required to be non-empty, and, even, if $A$ is infinite, when it is empty.

Now that we have an explicit representation of the free $\mathrm{CSP}(|,\Omega)$-monad in terms of $X$-processes, we indicate how to use it to give the semantics of the computational $\lambda$-calculus. First we need the structure of the monad. As we know from the above, the unit $\eta_X : X \to T_{\mathrm{CSP}(|,\Omega)}(X)$ is the map $x \mapsto (\{x\},\emptyset)$. Next, we need the homomorphic extension $g^\dagger : \mathscr{F}_{df}(X) \to \mathscr{F}_{df}(Y)$ of a given map $g : X \to \mathscr{F}_{df}(Y)$, i.e., the unique such homomorphism making the following diagram commute:




This is given by:

$$(g^\dagger(P))_{\mathscr{T}} = \{v \mid v \in T_P \cap A^*\} \cup \{vw \mid vx \in T_P,\ w \in g(x)_{\mathscr{T}}\}$$
$$(g^\dagger(P))_{\mathscr{R}} = \{(v,V) \in F_P\} \cup \{(vw,W) \mid vx \in T_P,\ (w,W) \in g(x)_{\mathscr{R}}\}$$

As regards the constructors and deconstructors, we have already given explicit representations of them as functions over (finitary) $X$-processes. We have also already given homomorphic treatments of the unary deconstructors. We finally give treatments of the binary deconstructors as unique solutions to equations, along similar lines to their treatment in the case of $\mathscr{F}_{df}$. Observe that:

$$(\textstyle\bigsqcap_{\vec{a}})_X(P_1,\ldots,P_n) = a_1P_1 \sqcap_X a_2P_2 \sqcap_X \ldots \sqcap_X a_nP_n$$
$$(\textstyle\bigsqcap^{\Omega}_{\vec{a}})_X(P_1,\ldots,P_n) = \Omega \sqcap_X a_1P_1 \sqcap_X a_2P_2 \sqcap_X \ldots \sqcap_X a_nP_n$$

Using this, one finds that $\sqcap_X$, $\sqcap_X^{\Omega,a_1\ldots a_n}$ and $\sqcap_X^{a_1\ldots a_n}$, the latter defined as in equation (9), are the unique functions which satisfy the evident analogues of equations (8) together with, making another use of the form of external choice made available by $\Omega$:

$$\eta(x) \sqcap P = \eta(x) \sqcap_X (\Omega \sqcap P)$$

and

$$(P_1,\ldots,P_n)\,\Box^{a_1\ldots a_n}\,\eta(x) = (\textstyle\bigsqcap_{\vec{a}}^{\Omega})_X(P_1,\ldots,P_n)\sqcap_X \eta(x)$$
$$(P_1,\ldots,P_n)\,\Box^{\Omega,a_1\ldots a_n}\,\eta(x) = (\textstyle\bigsqcap_{\vec{a}}^{\Omega})_X(P_1,\ldots,P_n)\sqcap_X \eta(x)$$

As regards concurrency, we define

$$||_{X,Y}\colon T_{\mathrm{CSP}(|,\Omega)}(X)\times T_{\mathrm{CSP}(|,\Omega)}(Y)\to T_{\mathrm{CSP}(|,\Omega)}(X\times Y)$$

together with functions

$$||^{a_1\ldots a_n}_{X,Y}\colon T_{\mathrm{CSP}(|,\Omega)}(X)^n\times T_{\mathrm{CSP}(|,\Omega)}(Y)\to T_{\mathrm{CSP}(|,\Omega)}(X\times Y)$$
$$||^{\Omega,a_1\ldots a_n}_{X,Y}\colon T_{\mathrm{CSP}(|,\Omega)}(X)^n\times T_{\mathrm{CSP}(|,\Omega)}(Y)\to T_{\mathrm{CSP}(|,\Omega)}(X\times Y)$$
$$||^{x}_{X,Y}\colon T_{\mathrm{CSP}(|,\Omega)}(Y)\to T_{\mathrm{CSP}(|,\Omega)}(X\times Y)$$

where the $a_i\in A$ are all different, and $x\in X$, by the analogues of equations (10) above, together with:

$$\begin{aligned}
\eta(x)\,||\,Q &= ||^x(Q)\\
\\
||^x(P\sqcap Q) &= ||^x(P)\sqcap\,||^x(Q)\\
||^x(\textstyle\bigsqcap_{i=1}^n a_iP_i) &= \Omega\\
||^x(\Omega\,\Box\,\textstyle\bigsqcap_{i=1}^n a_iP_i) &= \Omega\\
||^x(\eta(y)) &= \eta((x,y))\\
\\
(P_1,\ldots,P_n)\,||^{a_1\ldots a_n}\,\eta(x) &= \Omega\\
(P_1,\ldots,P_n)\,||^{\Omega,a_1\ldots a_n}\,\eta(x) &= \Omega
\end{aligned}$$

Much as before, the equations have a unique solution, with the $||$ component being $||_{X,Y}$.

As regards interleaving, we define

$$|||^l_{X,Y},|||^r_{X,Y}\colon T_{\mathrm{CSP}(|,\Omega)}(X)\times T_{\mathrm{CSP}(|,\Omega)}(Y)\to T_{\mathrm{CSP}(|,\Omega)}(X\times Y)$$

by:

$$\begin{aligned}
P\,|||^l_{\mathscr{T}_{df}(X,Y)}\,Q = {}&\{\varepsilon\}\cup\{w\mid u\in T_P\cap A^*,\ v\in T_Q\cap A^*,\ w\in u\,|^l\,v\}\cup\\
&\{w(x,y)\mid ux\in T_P,\ vy\in T_Q,\ w\in u\,|^l\,v\vee(u=v=w=\varepsilon)\}\\
\\
P\,|||^l_{\mathscr{R}_{df}(X,Y)}\,Q = {}&\{(\varepsilon,W)\mid(\varepsilon,W)\in F_P\}\cup\\
&\{(w,W)\mid(u,W)\in F_P,\ (v,W)\in F_Q,\ w\in u\,|^l\,v\}\\
\\
P\,|||^r_{X,Y}\,Q \quad = {}& Q\,|||^l_{Y,X}\,P
\end{aligned}$$

One has that:

$$P\,|||_{X,Y}\,Q = P\,|||^l_{X,Y}\,Q\,\Box\,P\,|||^r_{X,Y}\,Q$$

and that $|||^l_{X,Y}, |||^r_{X,Y}$ are components of the unique solutions to the analogues of equations (11) above, together with:

$$\begin{array}{ll} \eta(x) \,|||^l\, Q & = |||^{l,x}\,(Q) \\ \\ |||^{l,x}\,(P \sqcap Q) & = |||^{l,x}\,(P) \sqcap |||^{l,x}\,(Q) \\ |||^{l,x}\,(\bigsqcap_{i=1}^n a_i P_i) & = \Omega \\ |||^{l,x}\,(\Omega \sqcap \bigsqcap_{i=1}^n a_i P_i) & = \Omega \\ |||^{l,x}\,(\eta(y)) & = \eta(x,y) \end{array}$$

and corresponding equations for $|||^r$ and $|||^{r,y}$.

It would be interesting to check more completely which of the usual laws, as found in, e.g., [BHR84, Hoa85, DeN85], the CSP operators at the level of free $\mathrm{CSP}(|,\Omega)$-algebras obey. Note that some adjustments need to be made due to varying types. For example, $||$ is commutative, which here means that the following equation holds:

$$T_{\mathrm{CSP}(|,\Omega)}(\gamma_{X,Y})(P \,||_{X,Y}\, Q) = Q \,||_{Y,X}\, P$$

where $\gamma : X \times Y \to Y \times X$ is the commutativity map $(x,y) \mapsto (y,x)$.

## *7.1 Termination*

As remarked in the introduction, termination and sequencing are available in a standard way for terms of type `unit`. Syntactically, we regard `skip` as an abbreviation for $*$ and $M;N$ as one for $(\lambda x{:}\mathtt{unit}.N)(M)$ where $x$ does not occur free in $N$; semantically, we have a corresponding element of, and binary operator over, the free $\mathrm{CSP}(|,\Omega)$-algebra on the one-point set.

Let us use these ideas to treat CSP extended with termination and sequencing. We work with the finitary $\{\checkmark\}$-processes representation of $T_{\mathrm{CSP}(|,\Omega)}(\{\checkmark\})$. Then, following the above prescription, termination and sequencing are given by:

$$\mathtt{SKIP} = \{\checkmark\} \qquad\qquad P;Q = (x \in \{\checkmark\} \mapsto Q)^\dagger(P)$$

For general reasons, termination and sequencing, so-defined, form a monoid and sequencing commutes with all constructors in its first argument. For example we have that:

$$\bigsqcap_{i=1}^n a_i(P_i;Q) = (\bigsqcap_{i=1}^n a_i P_i);Q$$

Composition further commutes with $\sqcap$ in its second argument.

The deconstructors are defined as above except that in the case of the concurrency operators one has to adjust $||_{\{\checkmark\},\{\checkmark\}}$ and $|||_{\{\checkmark\},\{\checkmark\}}$ so that they remain within the world of the $\{\checkmark\}$-processes; this can be done by postcomposing them with the evident bijection between $\{\checkmark\} \times \{\checkmark\}$-processes and $\{\checkmark\}$-processes, and all this

restricts to the finitary processes. Alternatively one can directly consider these adjusted operators as deconstructors over the (finitary) $\{\surd\}$-processes.

The $\{\surd\}$-processes are essentially the elements of the stable failures model of [Ros98]. More precisely, one can define a bijection from Roscoe's model to our $\{\surd\}$-processes by setting $\theta(T,F) = (T,F')$ where

$$F' = \{(w,W) \in A^* \times \mathscr{P}(A) \mid (w, W \cup \{\surd\}) \in F\}$$

The inverse of $\theta$ sends $F'$ to the set:

$$\{(w,W), (w,W \cup \{\surd\}) \mid (w,W) \in F'\} \cup$$
$$\{(w,W) \mid w\surd \in T \wedge W \subseteq A\} \cup \{(w\surd, W) \mid w\surd \in T \wedge W \in A \cup \{\surd\}\}$$

and is a homomorphism between all our operators, whether constructors, deconstructors, termination, or sequencing (suitably defined), and the corresponding ones defined for Roscoe's model.

## 8 Discussion

We have shown the possibility of a principled combination of CSP and functional programming from the viewpoint of the algebraic theory of effects. The main missing ingredient is an algebraic treatment of binary deconstructors, although we were able to partially circumvent that by giving explicit definitions of them. Also missing are a logic for proving properties of these deconstructors, an operational semantics, and a treatment that includes recursion.

As regards a logic, it may prove possible to adapt the logical ideas of [PPr08, PPr09] to handle binary deconstructors; the main proof principle would then be that of *computation induction*, that if a proposition holds for all 'values' (i.e., elements of a given set $X$) and if it holds for the applications of each constructor to any given 'computations' (i.e., elements of $T(X)$) for which it is assumed to hold, then it holds for all computations. We do not anticipate any difficulty in giving an operational semantics for the above combination of the computational $\lambda$-calculus and CSP and proving an adequacy theorem.

To treat recursion algebraically, one passes from equational theories to inequational theories Th (inequations have the form $t \leq u$, for terms $t$, $u$ in a given signature $\Sigma$); inequational theories can include equations, regarding an equation as two evident inequations. There is a natural inequational logic for deducing consequences of the axioms: one simply drops symmetry from the logic for equations [Blo76]. Then $\Sigma$-algebras and Th-algebras are taken in the category of $\omega$-cpos and continuous functions, a free algebra monad always exists, just as in the case of sets, and the logic is complete for the class of such algebras. One includes a divergence constant $\Omega$ in the signature and the axiom

$$\Omega \leq x$$

so that Th-algebras always have a least element. Recursive definitions are then modelled by least fixed-points in the usual way. See [HPP06, Plo06] for some further explanations.

The three classical powerdomains: convex (aka Plotkin), lower (aka Hoare) and upper (aka Smyth) provide a useful illustration of these ideas [GHK03, HPP06]. One takes as signature a binary operation symbol $\sqcap$, to retain notational consistency with the present paper (a more neutral symbol, such as $\cup$, is normally used instead), and the constant $\Omega$; one takes the theory to be that $\sqcap$ is a semilattice (meaning, as before, that associativity, commutativity and idempotence hold) and that, as given above, $\Omega$ is the least element with respect to the ordering $\leq$. This gives an algebraic account of the convex powerdomain.

If one adds that $\Omega$ is the zero of the semilattice (which is equivalent, in the present context, to the inequation $x \leq x \sqcap y$) one obtains instead an algebraic account of the lower powerdomain. One then further has the notationally counterintuitive facts that $x \leq y$ is equivalent to $y \sqsubseteq x$, with $\sqsubseteq$ defined as in Section 3, and that $x \sqcap y$ is the supremum of $x$ and $y$ with respect to $\leq$; in models, $\leq$ typically corresponds to subset. It would be more natural in this case to use the dual order to $\sqsubseteq$ and to write $\sqcup$ instead of $\sqcap$, when we would be dealing with a join-semilattice with a least element whose order coincides with $\leq$.

If one adds instead that $x \sqcap y \leq x$, one obtains an algebraic account of the upper powerdomain. One now has that $x \leq y$ is equivalent in this context to $x \sqsubseteq y$, that $x \sqcap y$ is the greatest lower bound of $x$ and $y$, and that $x \sqcap \Omega = \Omega$ (but this latter fact is not equivalent in inequational logic to $x \sqcap y \leq x$); in models, $\leq$ typically corresponds to superset. The notations $\sqcap$ and $\sqsubseteq$ are therefore more intuitive in the upper case, and there one has a meet-semilattice with a least element whose order coincides with $\leq$.

It will be clear from these considerations that the stable failures model fits into the pattern of the lower powerdomain and that the failures/divergences model fits into the pattern of the upper powerdomain. In the case of the stable failures model it is natural, in the light of the above considerations, to take Th to be $\mathrm{CSP}(|, \Omega)$ together with the axiom $\Omega \leq x$. The $X$-processes with countably many traces presumably form the free algebra over $X$, considered as a discrete $\omega$-cpo; one should also characterise more general cases than discrete $\omega$-cpos.

One should also investigate whether a fragment of the failures/divergences model forms the initial model of an appropriate theory, and look at the free models of such a theory. The theory might well be found by analogy with our work on the stable failures model, substituting (12) for (13) and, perhaps, using the mixed-choice constructor, defined below, to overcome any difficulties with the deconstructors. One would expect the initial model to contain only finitely-generable processes, meaning those which, at any trace, either branch finitely or diverge (and see the discussion in [Ros98]).

Our initial division of our selection of CSP operators into constructors and deconstructors was natural, although it turned out that a somewhat different division, with 'restricted' constructors, resulted in what seemed to be a better analysis (we were not able to rule out the possibility that there are alternative, indirect, definitions of the deconstructors with the original choice of constructors). One of these

restricted constructors was a deterministic choice operator making use of the divergence constant $\Omega$. There should surely, however, also be a development without divergence that allows the interpretation of the combination of CSP and functional programming.

We were, however, not able to do this using CSP($|$): the free algebra does not seem to support a suitable definition of concealment, whether defined directly or via a homomorphism. For example a straightforward extension of the homomorphic treatment of concealment in the case of the initial algebra (cf. Section 5) would give

$$(a.\underline{x} \Box b.\mathtt{Stop})\backslash a = \underline{x} \sqcap (\underline{x} \Box b.\mathtt{Stop})$$

However, our approach requires the right-hand side to be equivalent to a term built from constructors only, but no natural candidates came forward—all choices that came to mind lead to unwanted identifications.

We conjecture that, taking instead, as constructor, a *mixed-choice* operator of the form:

$$\Box_i \alpha_i.x_i$$

where each $\alpha_i$ is either an action or $\tau$, would lead to a satisfactory theory. This new operator is given by the equation:

$$\Box_i \alpha_i.x_i = \sqcap_{\alpha_i=\tau} x_i \sqcap \left( \Box_{\alpha_i=\tau} x_i \Box \Box_{\alpha_i\neq\tau} \alpha_i.x_i \right)$$

and there is a homomorphic relationship with concealment:

$$(\Box_i \alpha_i.x_i)\backslash a = \Box_i (\alpha_i\backslash a).(x_i\backslash a)$$

(with the evident understanding of $\alpha_i\backslash a$). Note that in the stable failures model we have the equation:

$$\Box_i \alpha_i.x_i = \sqcap_{\alpha_i=\tau} x_i \sqcap \left( \Omega \Box \Box_{\alpha_i\neq\tau} \alpha_i.x_i \right)$$

which is presumably why the deterministic choice operator available in the presence of $\Omega$ played so central a rôle there.

In a different direction, one might also ask if there is some problem if we alternatively take an extended set of operators as constructors. For example, why not add relabelling with its equations to the axioms? As the axioms inductively determine relabelling on the finitary refusal sets model, that would still be the initial algebra, and the same holds if we add any of the other operators we have taken as deconstructors.

However, the $X$-refusal sets would not longer be the free algebra, as there would be extra elements, such as $f(x)$ for $x \in X$, where $f$ is a relabelling function. We would also get some undesired equations holding between terms of the computational $\lambda$-calculus. For any $n$-ary constructor op and evaluation context $E[-]$, one has in the monadic semantics:

$$E[\mathrm{op}(M_1,\ldots,M_n)] = \mathrm{op}(E[M_1],\ldots,E[M_n])$$

So one would have $E[f(M)] = f(E[M])$ if one took relabelling as a constructor, and, as another example, one would have $E[M \mid\mid N] = E[M] \mid\mid E[N]$ if one took the concurrency operator as a constructor.

It will be clear to the reader that, in principle, one can investigate other process calculi and their combination with functional programming in a similar way. For example for Milner's CCS [Mil80] one could take action prefix (with names, conames and $\tau$) together with `NIL` and the sum operator as constructors, and as axioms that we have a semilattice with a zero, for strong bisimulation, together with the usual $\tau$-laws, if we additionally wish to consider weak bisimulation. The deconstructors would be renaming, hiding, and parallel, and all should have suitable polymorphic versions in the functional programming context. Other process calculi such as the $\pi$-calculus [SW03, Sta08], or even the stochastic $\pi$-calculus [Pri95, KS08], might be dealt with similarly. In much the same way, one could combine parallelism with a global store with functional programming, following the algebraic account of the resumptions monad [HPP06, AP09] where the constructors are the two standard ones for global store [PP02], a nondeterministic choice operation, and a unary 'suspension' operation.

A well-known feature of the monadic approach [HPP06] is that it is often possible to combine different effects in a modular way. For example, the global side-effects monad is $(S \times -)^S$ where $S$ is a suitable set of states. A common combination of it with another monad $T$ is the monad $T(S \times -)^S$. So, taking $T = T_{\mathrm{CSP}(|)}$, for example, we get a combination of CSP with global side-effects.

As another example, given a monoid $M$, one has the $M$-action monad $M \times -$ which supports a unary $M$-action effect constructor $m.-$, parameterised by elements $m$ of the monoid. One might use this monad to model the passage of time, taking $M$ to be, for example, the monoid of the natural numbers $\mathbb{N}$ under addition. A suitable combination of this monad with ones for CSP may yield helpful analyses of timed CSP [RR99, OS06], with $\mathit{Wait}\ n; -$ given by the $\mathbb{N}$-action effect constructor. We therefore have a very rich space of possible combinations of process calculi, functional programming and other effects, and we hope that some of these prove useful.

Finally, we note that there is no general account of how the equations used in the algebraic theory of effects arise. In such cases as global state, nondeterminism or probability, there are natural axioms and monads already available, and it is encouraging that the two are equivalent [PP02, HPP06]. One could investigate using operational methods and behavioural equivalences to determine the equations, and it would be interesting to do so. Another approach is the use of 'test alge-

bras' [SS06, KP09]. In the case of process calculi one naturally uses operational methods; however the resulting axioms may not be very modular, or very natural mathematically, and, all in all, in this respect the situation is not satisfactory.

## References


AP09. M. Abadi & G. D. Plotkin, A model of cooperative threads, *Proc. POPL 2009* (eds. Z. Shao & B. C. Pierce), ACM Press, 29–40, 2009.

AGM95. S. Abramsky, D. M. Gabbay & T. S. E. Maibaum (eds), *Handbook of Logic in Computer Science (Vol. 1), Background: Mathematical Structures*, Oxford University Press, 1995.

BHM02. N. Benton, J. Hughes & E. Moggi, Monads and effects, *Proc. APPSEM 2000*, LNCS **2395**, 42–122, Springer, 2002.

BK85. J. A. Bergstra & J. W. Klop, Algebra of communicating processes with abstraction, *Theor. Comput. Sci.* **37**, 77–121, 1985.

BK86. J. A. Bergstra & J. W. Klop, Algebra of communicating processes, *Proc. of the CWI Symp. Math. and Comp. Sci.* (eds. J. W. de Bakker, M. Hazewinkel & J. K. Lenstra), 89–138, North-Holland, 1986.

BKO87. J. A. Bergstra, J. W. Klop, & E.-R. Olderog, Failures without chaos: a new process semantics for fair abstraction. *Proc. of the* $3^{th}$ *IFIP WG 2.2 working conference on Formal Description of Programming Concepts* (ed. M. Wirsing), 77–103, North-Holland, 1987.

Blo76. S. L. Bloom, Varieties of ordered algebras, *J. Comput. Syst. Sci.*, **13**(2), 200–212, 1976.

Bor94. F. Borceux, *Handbook of Categorical Algebra 2*, Encyclopedia of Mathematics and its Applications **51**, Cambridge University Press, 1994.

BHR84. S. D. Brookes, C. A. R. Hoare & A. W. Roscoe, A theory of communicating sequential processes, *J. ACM* **31**(3), 560–599, 1984.

DeN85. R. De Nicola, Two complete axiom systems for a theory of communicating sequential processes, *Information and Control* **64**, 136–172, 1985.

EFG08. K. Ebrahimi-Fard & L. Guo, Rota-Baxter algebras and dendriform algebras, *Journal of Pure and Applied Algebra* **212** (2), 320–33, 2008.

GHK03. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove & D. S. Scott, *Continuous Lattices and Domains*, Encyclopedia of Mathematics and its Applications **93**, Cambridge University Press, 2003.

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

HPP06. J. M. E. Hyland, G. D. Plotkin & A. J. Power, Combining effects: sum and tensor, Clifford Lectures and the Mathematical Foundations of Programming Semantics, (eds. S. Artemov and M. Mislove), *Theor. Comput. Sci.* **357**(1–3), 70–99, 2006.

KP09. K. Keimel & G. D. Plotkin, Predicate transformers for extended probability and nondeterminism, *Mathematical Structures in Computer Science* **19**(3), 501–539, Cambridge University Press, 2009.

KS08. B. Klin & V. Sassone, Structural operational semantics for stochastic process calculi, *Proc. 11th. FoSSaCS* (ed. R. M. Amadio), LNCS **4962**, 428–442, Springer, 2008.

Mil80. A. J. R. G. Milner, *A Calculus of Communicating Systems*, Springer, 1980.

Mog89. E. Moggi, Computational lambda-calculus and monads, *Proc. 3rd. LICS*, 14–23, IEEE Press, 1989.

Mog91. E. Moggi, Notions of computation and monads, *Inf. & Comp.* **93**(1), 55–92, 1991.

OS06. J. Ouaknine & S. Schneider, Timed CSP: a retrospective, *Proceedings of the Workshop "Essays on Algebraic Process Calculi" (APC 25)*, Electr. Notes Theor. Comput. Sci., **162**, 273–276, 2006.

Plo06. G. D. Plotkin, Some varieties of equational logic, *Essays Dedicated to Joseph A. Goguen* (eds. K. Futatsugi, J.-P. Jouannaud & J. Meseguer), LNCS **4060**, 150–156, Springer, 2006.
PP02. G. D. Plotkin & A. J. Power, Notions of computation determine monads, *Proc. 5th. FOSSACS*, LNCS **2303**, 342–356, Springer, 2002.
PP04. G. D. Plotkin & A. J. Power, Computational effects and operations: an overview, *Proc. Workshop on Domains VI* (eds. M. Escardó and A. Jung), *Electr. Notes Theor. Comput. Sci.* **73**, 149–163, Elsevier, 2004.
PPr08. G. D. Plotkin & M. Pretnar, A logic for algebraic effects, *Proc. 23rd. LICS*, 118-129, IEEE Press, 2008.
PPr09. G. D. Plotkin & M. Pretnar, Handlers of algebraic effects, *Proc. 18th. ESOP*, 80–94, 2009.
Pri95. C. Priami, Stochastic pi-calculus, *Comput. J.* **38** (7), 578–589, 1995.
RR99. G. M. Reed & A. W. Roscoe, The timed failures-stability model for CSP, *Theor. Comput. Sci.* **211** (1-2), 85-127, 1999.
Ros94. A. W. Roscoe, Model-checking CSP, *A Classical Mind: Essays in Honour of C. A. R. Hoare* (ed. A. W. Roscoe), 353–337, Prentice-Hall, 1994.
Ros98. A. W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, 1998.
SW03. D. Sangiorgi & D. Walker, *The $\pi$-Calculus: A Theory of Mobile Processes*, Cambridge University Press, 2003.
Sca. B. Scattergood, *The Semantics and Implementation of Machine-Readable CSP*, D.Phil Thesis, Oxford University, 1998.
SS06. M. Schröder & A. Simpson, Probabilistic observations and valuations (extended abstract), *Electr. Notes Theor. Comput. Sci.* **155**, 605–615, 2006.
Sta08. I. Stark, Free-algebra models for the pi -calculus, *Theor. Comput. Sci.* **390**(2-3), 248–270, 2008.

## Appendix: The computational $\lambda$-calculus

In this appendix, we sketch (a slight variant of) the syntax and semantics of Moggi's computational $\lambda$-calculus, or $\lambda_c$-calculus [Mog89, Mog91]. It has types given by:

$$\sigma ::= \mathtt{b} \mid \mathtt{unit} \mid \sigma \times \sigma \mid \mathtt{empty} \mid \sigma \to \sigma$$

where $\mathtt{b}$ ranges over a given set of base types, e.g., $\mathtt{nat}$; the type construction $T\sigma$ may be defined to be $\mathtt{unit} \to \sigma$. The terms of the $\lambda_c$-calculus are given by:

$$M ::= x \mid g(M) \mid * \mid \mathtt{in}M \mid (M,M) \mid \mathtt{fst}M \mid \mathtt{snd}M \mid \lambda x{:}\sigma.M \mid MM$$

where $g$ ranges over given unary function symbols of given types $\sigma \to \tau$, such as $0 : \mathtt{unit} \to \mathtt{nat}$ or $\mathtt{succ} : \mathtt{nat} \to \mathtt{nat}$, if we want the natural numbers, or $\mathrm{op} : T(\sigma) \times \ldots \times T(\sigma) \to T(\sigma)$ for some operation symbol from a theory for which $T$ is the free algebra monad. There are standard notions of free and bound variables and of closed terms and substitution; there are also standard typing rules for judgements $\Gamma \vdash M : \sigma$, that the term $M$ has type $\sigma$ in the context $\Gamma$ (contexts have the form $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$), including:

$$\frac{\Gamma \vdash M : \mathtt{empty}}{\Gamma \vdash \mathtt{in}M : \sigma}$$

A $\lambda_c$-model (on the category of sets—Moggi worked more generally) consists of a monad $T$, together with enough information to interpret basic types and the given function symbols. So there is a given set $[\![b]\!]$ to interpret each basic type $b$, and then every type $\sigma$ receives an interpretation as a set $[\![\sigma]\!]$; for example $[\![\mathtt{empty}]\!] = \emptyset$. There is also given a map $[\![\sigma]\!] \to T([\![\tau]\!])$ to interpret every given unary function symbol $g : \sigma \to \tau$. A term $\Gamma \vdash M : \sigma$ of type $\sigma$ in context $\Gamma$ is modelled by a map $[\![M]\!] : [\![\Gamma]\!] \to T[\![\sigma]\!]$ (where $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n]\!] = [\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!]$). For example, if $\Gamma \vdash \mathtt{in}M : \sigma$ then $[\![\mathtt{in}M]\!] = 0_{[\![\sigma]\!]} \circ [\![M]\!]$ (where, for any set $X$, $0_X$ is the unique map from $\emptyset$ to $X$).

We define values and evaluation contexts. Values can be thought of as (syntax for) completed computations, and are defined by:

$$V ::= x \mid * \mid (V,V) \mid \mathtt{in}V \mid \lambda x : \sigma . M$$

together with clauses such as:

$$V ::= 0 \mid \mathtt{succ}(V)$$

depending on the choice of basic types and given function symbols. We may then define evaluation contexts by:

$$E ::= [-] \mid \mathtt{in}E \mid (E,M) \mid (V,E) \mid EM \mid VE \mid \mathtt{fst}(E) \mid \mathtt{snd}(E)$$

together with clauses such as:

$$E ::= \mathtt{succ}(E)$$

depending on the choice of basic types and given function symbols. We write $E[M]$ for the term obtained by replacing the 'hole' $[-]$ in an evaluation term $E$ by a term $M$. The computational thought behind evaluation contexts is that in a program of the form $E[M]$, the first computational step arises within $M$.