

Unfolding CSP

Mikkel Bundgaard and Robin Milner

Dedicated to Sir Tony Hoare for his 75th birthday

Appreciation from the second author Tony Hoare and I have exchanged ideas on concurrent processes for three decades. To a sufficiently distant observer it has seemed that we were doing the same thing, and that therefore we should not have made it look different. A closer view shows this to be false. Complementary things, yes; and both have been enriched by the cross-fertilisation.

A little history shows something of these different approaches. Around 1979 we first discovered our complementary interests in concurrency. Tony at first expressed his ideas through the medium of a programming language [6], and I through a prototypical algebraic theory [8]. The difference became plainer as time went on. Tony was keen to find a single formalism in which specifications of concurrent systems could be refined into programs. I, on the other hand, was keen to find a mathematical concept of *process* that could stand in analogy with the familiar notion of (single valued) *function*, and I was happy that specification should be done in an associated logic. We enjoyed discussing these things. I recall one discussion at a blackboard (or was it a whiteboard?) where Tony hinted to me his first ideas about failures semantics.

The present short paper, in honour of Tony and continuing our long, friendly and sometimes rivalrous collaboration, is another step towards harmonising our approaches.

1 Motivation

Over the past decade a topographical process model called *Bigraphs* [11] has been developed, aiming to provide a rigorous definitional platform for ubiquitous computing. A first priority has been to show how existing process calculi can be embedded faithfully in the bigraphical framework. This effort has been largely successful,

Mikkel Bundgaard
IT University of Copenhagen, Rued Langgaards Vej 7, e-mail: mikkelbu@itu.dk

Robin Milner
University of Cambridge, 15 JJ Thomson Avenue, e-mail: rm135@cam.ac.uk

and the effort on bigraphs is now broadening into the experimental modelling of real ubiquitous systems.

An exception to this success has, hitherto, been the attempt to accommodate CSP [7] within the bigraphical framework. Why has this seemed difficult? (Readers not familiar with bigraphs can safely skim, or even ignore, the rest of this section, since bigraphs are hardly mentioned thereafter.)

The difficulty is not due to the semantic treatment. CSP favours a preorder based upon the notions of refusals and failures, thus providing process specifications as well as fully defined processes. On the other hand, CCS [9] emphasises the equivalence relation of bisimilarity, relying on other means, such as an associated logic, to provide specification. However, this difference is no barrier to accommodating CSP-style specifications in the bigraphical framework, where both semantic treatments fit comfortably.

More significant are two features special to CSP: channel ownership, and a large repertoire of operations on processes. Consider the latter first. The bigraph model is a symmetric partial monoidal (spm) category, and the main operations of parallel composition in both CCS and CSP can be modelled essentially by the tensor product, which is a structural operation central to such categories. However, it was not clear how to model other CSP operations, such as general choice or interleaving. It seemed that a more cumbersome categorical structure would be needed, and this would not do justice to the inherent simplicity of CSP. But the appearance is unfounded; by the device of *unfolding*—a special case of parametric recursion—these operations (many, perhaps all) can be defined by reduction to a normal form, a device present in the bigraph model.

Turning to channel ownership: In CSP, one can maintain the distinction among the channel-sets owned by several processes by careful limitation of the use of non-injective renaming; for example, in his book [7] Hoare uses process labelling. In bigraphs it appears that similar control can be exercised by careful restriction of the use of substitutions, which are themselves elementary bigraphs. This is achieved by defining a so-called *sorting*, a device inspired by many-sorted algebra to determine a sub-spm-category of bigraphs for any particular application.

In a future paper we shall identify the sorted bigraphical model for CSP. Here we confine ourselves to the representation of CSP operators by unfolding, which can be presented independently of bigraphs.

2 Outline and Disclaimer

As we have already declared, we hope in a future paper to represent CSP as a bigraphical reactive system. Here, we wish to demonstrate the complementarity between the CSP approach and one that features bisimilarity. We avoid any judgement that one or the other approach is superior. Our main focus here is upon finding how the richly-various set of CSP operators can be defined so as to harmonise with bisimilarity.

Let us make explicit the differing concerns of the two approaches. CSP defines the *failures* preorder that allows specifications, as well as their refinement into implementations, to be expressed within a single syntactic framework. The emphasis is upon refinement. On the other hand, the aim in CCS [9] is to characterize the notion of *process* as an equivalence class of process expressions, respecting the order in which non-deterministic choices are made as the process progresses. This equivalence relation is the (strong) bisimilarity \sim introduced by Park [12], an improvement on the equivalence originally given for CCS.

Bisimilarity is a congruence, i.e. preserved by syntactic context. Furthermore bisimilarity is also preserved by the transition relation between processes. This latter property—preservation by transition—is not satisfied by the CSP failures preorder or its variants; it could not be, since these preorders allow a specification to be refined into process expressions that differ in when they make non-deterministic choices. To achieve complementarity between these approaches we must, as far as possible, define any CSP operator OP to ‘make sense’ for bisimilarity; i.e. (for binary OP) if $P_1 \sim Q_1$ and $P_2 \sim Q_2$ then $OP(P_1, P_2) \sim OP(Q_1, Q_2)$. We believe that this can be achieved for all the operators; here we shall ensure it for a selection of them. Moreover, the equational laws of CSP declared in Hoare’s book for the operators we examine (except those involving \square , non-deterministic ‘or’) are almost all satisfied not just for failures equivalence, but for strong bisimilarity.

Now, since the failures preorder includes strong bisimilarity, the preorder can be understood to be over processes themselves, not only over the expressions that denote them. It is therefore possible to combine the theories of CCS and CSP.

To define the CSP operators over processes, we shall work with a new head normal form. This will be based upon a generalisation of the CSP choice construction. We can then define the operators quite simply, using the *unfolding* of process expressions. This notion of unfolding (related to that already studied for bigraphs, see [11]) turns out to be confluent. Moreover, the equivalence induced by unfolding is closely related to bisimilarity, even coinciding with it in the absence of recursively defined processes.

Roscoe [15] has already defined a head normal form for CSP, and thence a full normal form following De Nicola [5]. These forms depend crucially upon the way \square (non-deterministic ‘or’) distributes over other operators. Thus, at least for finite processes (those without recursion), two process expressions in normal form are essentially identical if and only if they are *failures-equivalent*.¹ However, our purpose is to achieve a similar result for *bisimilarity equivalence*, where we regard non-deterministic ‘or’ as an action that does not in general commute with other actions. We therefore unfold the \square operator in the same way as the other operators.

Let us repeat: the failures model and the notion of bisimilarity are not in competition; each has a rationale with independent value.

¹ Roscoe’s normal form also caters explicitly for divergence; this lies beyond the scope of our paper.

3 CSP with Enriched Syntax

Recall the well-known CSP syntax, employing several binary operators. The constructions were designed to introduce the phenomena of choice, concurrency, non-determinism, hiding, etc., one-by-one.

Channel names x, y, \dots (also called channels) are drawn from an infinite vocabulary \mathcal{X} . Each process P has a finite² alphabet $\alpha P \subset \mathcal{X}$. CSP has many process constructions, often binary operators. We shall not model them all here; for example, we shall not consider the sequential composition $P; Q$ nor the interrupt operator $P \triangle Q$. We consider the following six process constructions, and conjecture that our treatment can be extended to some or all of the others:

P, Q	$::=$		
	$(x_1 \rightarrow P_1 \mid \dots \mid x_n \rightarrow P_n)$	choice	$\alpha P = \alpha P_i \supseteq \{x_1, \dots, x_n\}$ (x_i distinct)
	$P_1 \parallel P_2$	parallel	$\alpha P = \alpha P_1 \cup \alpha P_2$
	$Q \setminus Y$	hiding	$\alpha P = \alpha Q \setminus Y$
	$P_1 \square P_2$	general choice	$\alpha P = \alpha P_1 = \alpha P_2$
	$P_1 \parallel\!\!\parallel P_2$	interleave	$\alpha P = \alpha P_1 = \alpha P_2$
	$P_1 \sqcap P_2$	‘or’	$\alpha P = \alpha P_1 = \alpha P_2$

The n -ary choice construction is deterministic, in the sense that there is a unique choice of action for each channel $x \in \alpha P$.

The important intuition of parallel composition $P \parallel Q$ is that on the shared channels $x \in \alpha P \cap \alpha Q$ the actions of P and Q are synchronised, while on unshared channels only the unique possessor acts, the other remain unchanged. Thus the parallel composition preserves determinism.

The other operators introduce non-determinism in various ways. In particular, the hiding construction $Q \setminus Y$ renders the actions in $\alpha Q \cap Y$ unobservable. Hence the actions are able to occur without further participants.

In his book, Hoare avoids formalising the operational behaviour of CSP. It has been formalised by others, probably first by Brookes, Roscoe and Walker [3], employing the structural operational semantics pioneered by Plotkin [13]. However we adopt a different approach here, defining the behaviour by unfolding each syntactic construction to a *head normal form*. As we shall see, this unfolding is a confluent reduction system. Indeed, for finite CSP (i.e. no user-defined recursions) this unfolding terminates in a *normal form*; moreover, two process terms unfold to the same normal form if and only if they are strongly bisimilar.

In the next section we shall define how each of the several operators unfolds. But first we have to define our head normal forms. To understand our definition better, it is helpful to recall an equation that records how parallel composition ‘unfolds’ into a choice among alternatives. It was first introduced as a law [8], and later became the expansion theorem of CCS [9]. Recall that the parallel composition of two CCS processes is written $P \mid Q$, and that this allows P and Q each to perform an input

² For simplicity, we assume alphabets to be finite. Later work may relax this restriction.

action x or output action \bar{x} independently or to communicate—creating a hidden action τ —if P can do x and Q do \bar{x} or vice-versa. Thus interaction in CCS involves exactly two participants, unlike in CSP.

The expansion theorem then captures the behaviour of the parallel composition of two ‘summations’ (or choice forms, in CSP terminology), expressing it again as a summation. Each summand takes the form $\mu.P$, where the prefix μ takes the form x , \bar{x} or τ . (A summand $\mu.P$ corresponds to a choice element $\mu \rightarrow P$ in CSP.) We declare that $\bar{\bar{x}} = x$. Letting $P = \sum_i \lambda_i.P_i$ and $Q = \sum_j \mu_j.Q_j$ the expansion theorem asserts:

$$\sum_i \lambda_i.P_i \mid \sum_j \mu_j.Q_j \sim \sum_i \lambda_i.(P_i \mid Q) + \sum_j \mu_j.(P \mid Q_j) + \sum_{\lambda_i = \bar{\mu}_j} \tau.(P_i \mid Q_j)$$

where \sim denotes strong equivalence, later improved to strong bisimilarity by David Park [12].

This theorem arose from the semantics of CCS defined by structured operational semantics [13], i.e. by an inference system over labelled transitions. But it suggests another way to formulate the semantics. Let us convert the expansion theorem into a rewriting rule

$$\sum_i \lambda_i.P_i \mid \sum_j \mu_j.Q_j \hookrightarrow \sum_i \lambda_i.(P_i \mid Q) + \sum_j \mu_j.(P \mid Q_j) + \sum_{\lambda_i = \bar{\mu}_j} \tau.(P_i \mid Q_j)$$

where the directed relation \hookrightarrow unfolds a parallel composition into a summation. CCS would also need such an unfolding rule for restriction (hiding); then bisimilarity, and other semantic equivalences or preorders, can be defined in terms of such rules.

We propose to adopt this approach in the bigraph model; it will be especially useful for process calculi like CSP which have many operators, each giving rise to an unfolding rule. We shall do this here for CSP directly, in terms of its familiar syntax. Thus we avoid further reference to bigraphs, but we pave the way for a simple embedding of CSP in bigraphs, allowing the insights of CSP to be conferred upon other calculi.

For this to work, we must enrich the choice construction

$$P = (\mu_1 \rightarrow P_1 \mid \cdots \mid \mu_n \rightarrow P_n)$$

to allow non-determinism; this entails allowing the actions μ_i to be not necessarily distinct, and also to be drawn from $\alpha P \cup \{\tau\}$, where τ represents a hidden action. For simplicity, since the order of choices is immaterial, we shall define Procsets_X to mean the set of finite³ sets of process expressions over alphabet X , and enrich the choice construction to a so-called *head normal form* (hnf):

$$P = F : \alpha P \cup \{\tau\} \longrightarrow \text{Procsets}_{\alpha P} .$$

³ Here again, it may be possible to remove the finiteness constraint for some purposes.

Thus a hnf F is a function mapping each action $\mu \in \alpha P \cup \{\tau\}$ to a finite set of possible successor processes. For example, the hnf

$$F : x \mapsto \{P_1, P_2\}, y \mapsto \{Q\}, \tau \mapsto \{R_1, R_2\}$$

could be written in an extension of the CSP choice notation as

$$F = (x \rightarrow P_1 \mid x \rightarrow P_2 \mid y \rightarrow Q \mid \tau \rightarrow R_1 \mid \tau \rightarrow R_2) .$$

We conjecture that sequential composition $P;Q$ can be treated by introducing a special hnf SKIP_X , for each alphabet X , to represent successful termination. We define a process P to be in *normal form* if and only if the only constructions in P are enriched choices.

Definition 1 (normal form). A process P is a *normal form* iff the only process operators in P are enriched choices.

In the rest of the paper we will present several relations on CSP terms, and we will tacitly assume that these relations only relate terms with the same alphabet.

4 Unfolding

We now define all the CSP operators listed above by means of unfolding to hnfs. Let us use F, G, H to range over hnfs, P, Q, R over arbitrary processes, and S, T over finite sets of processes. Furthermore we extend the operators to sets of processes by defining, for example,

$$P \parallel S \stackrel{\text{def}}{=} \{P \parallel Q : Q \in S\} \quad \text{and} \quad S \parallel T \stackrel{\text{def}}{=} \{P \parallel Q : P \in S, Q \in T\} .$$

For each operator we give a single axiom that defines its unfolding for arguments that are themselves hnfs. In the following we assume $\alpha F = X$ and $\alpha G = Y$, and to avoid too many parentheses we assume that set union \cup binds less tightly than all the operators. Note that $X = Y$ in the case of the operators \square (general choice), \parallel (interleaving), and \sqcap (or).

$$F \parallel G \hookrightarrow H \text{ where } H(z) = \begin{cases} F(z) \parallel G & (z \in X \setminus Y) \\ F \parallel G(z) & (z \in Y \setminus X) \\ F(z) \parallel G(z) & (z \in X \cap Y) \end{cases}$$

$$\text{and } H(\tau) = F(\tau) \parallel G \cup F \parallel G(\tau)$$

$$F \sqcap G \hookrightarrow H \text{ where } H(x) = F(x) \cup G(x) \quad (x \in X)$$

$$\text{and } H(\tau) = F(\tau) \sqcap G \cup F \sqcap G(\tau)$$

$$F \parallel\!\!\parallel G \hookrightarrow H \text{ where } H(\mu) = F(\mu) \parallel\!\!\parallel G \cup F \parallel\!\!\parallel G(\mu) \quad (\mu \in X \cup \{\tau\})$$

$$F \setminus Z \hookrightarrow H \text{ where } H(x) = F(x) \setminus Z \quad (x \in X \setminus Z)$$

$$\text{and } H(\tau) = F(\tau) \setminus Z \cup \bigcup_{x \in X \cap Z} F(x) \setminus Z$$

$$F \sqcap G \hookrightarrow H \text{ where } H(\mu) = F(\mu) \cup G(\mu) \quad (\mu \in X \cup \{\tau\})$$

In addition, we allow unfolding to occur in any context; that is:

$$\text{for any process context } \mathcal{C}[\cdot], \text{ if } P \hookrightarrow P' \text{ then } \mathcal{C}[P] \hookrightarrow \mathcal{C}[P'] .$$

For finite CSP, this defines unfolding fully. Infinite CSP allows mutually recursive definitions of any number of process identifiers. For each such identifier A there is a unique defining expression P_A , and the definition augments the unfolding operation with the axiom

$$A \hookrightarrow P_A .$$

Each P_A may contain occurrences of any of the process identifiers, but these occurrences must be guarded—i.e. must be within the body of some enriched choice construction.

Now recall that a (directed) relation \rightarrow is *confluent* if, whenever $a \rightarrow^* a_1$ and $a \rightarrow^* a_2$, then there exists a' such that $a_1 \rightarrow^* a'$ and $a_2 \rightarrow^* a'$.

Theorem 1 (confluence). *The unfolding relation \hookrightarrow is confluent.*

Proof. The proof will be given fully in a later paper. Intuitively, it relies on the intuition that unfolding loses none of the non-determinism represented in each operation.

In summary, the proof uses a generalisation of the *Parallel Moves Lemma* for term rewriting, which may be found as Lemma 4.3.3 in [17]. This asserts that a reduction relation on terms is confluent provided its rules are left-linear (no repeated parameter variables on the left-hand side of a rule) and non-overlapping (no preemption of one rule by another). These conditions are satisfied by our set of unfolding rules. The Lemma is slightly generalised since it originally applies to terms, whose syntax consists of strings of symbols, and we have introduced the syntax of enriched choice constructions, which are finite maps to sets. \square

Although unfolding is confluent, it may fail to terminate. One reason is the recursive unfolding of process identifiers; indeed, the single recursive rule $A \hookrightarrow \{x \mapsto \{A\}\}$

repeats the action x ad infinitum. Perhaps surprisingly, this is the only reason for non-termination. To see this, we first define strong normalisation.

Definition 2 (strong normalisation). A process P is *strongly normalising* if it has no infinite unfolding.

Theorem 2 (strong normalisation). *Recursion-free processes are strongly normalising.*

Proof. In outline, we assign to each term a multiset, consisting of the heights of all the process operators in the term, except for enriched choice operators. For instance, the term

$$((F \parallel F) \parallel F) \square ((F \parallel F) \parallel F)$$

—where F is an empty hnf—will be assigned the multiset $\{2, 2, 3, 3, 4\}$, since there are two operator occurrences (both \parallel) with height 2, two (both $\parallel\parallel$) with height 3, and one (\square) with height 4. (For convenience, leaves have height 1).

Now it is known [4] that if an ordering on a set C is well-founded, then so is its extension to an ordering on finite multisets over C . Here, take C to be the natural numbers; then the ordering extends to finite multisets as follows: $B \succ B'$ if B' can be obtained from B by replacing some elements by finitely many smaller elements.

It remains to prove that every unfolding strictly decreases this ordering. Indeed, an unfolding does not increase the height of a term; and it replaces a single process operator with a finite number of process operators, but of smaller height. \square

For the next section we shall need to use the transitive reflexive closure of unfolding, written \hookrightarrow^* . We shall also need the transitive reflexive symmetric closure of unfolding which we will call structural congruence.

Definition 3 (structural congruence). *Structural congruence*, denoted by \equiv , is the transitive reflexive symmetric closure of unfolding, $(\hookrightarrow \cup \hookleftarrow)^*$.

This concludes the properties of unfolding that we need.

5 Bisimilarity

We shall now introduce transition relations labelled by actions $\mathcal{X} \cup \{\tau\}$; we shall let μ range over these labels. The simplest case is for a hnf F ; we want $F \searrow_{\mu} P'$ whenever $P' \in F(\mu)$. However, we want transitions for an arbitrary process term P , not necessarily a hnf; so we allow P to unfold into a hnf first. Thus transitions are defined by the following rule:

$$\frac{P \hookrightarrow^* F \quad P' \in F(\mu)}{P \searrow_{\mu} P'}$$

Having defined transitions, we can now define strong bisimilarity in the usual way.

Definition 4 (bisimulation, bisimilarity). A strong bisimulation is a binary relation \mathcal{R} over processes such that, whenever $(P, Q) \in \mathcal{R}$ and $P \searrow_{\mu} P'$ then there exists Q' such that $Q \searrow_{\mu} Q'$ and $(P', Q') \in \mathcal{R}$, and conversely when $Q \searrow_{\mu} Q'$ then there exists P' such that $P \searrow_{\mu} P'$ and $(P', Q') \in \mathcal{R}$. Bisimilarity, denoted by \sim , is the largest bisimulation; that is, it is the union of all bisimulations.

A weaker equivalence relation, *weak bisimilarity*, places less constraint upon τ actions $P \searrow_{\tau} P'$; for example, two τ actions in sequence are equivalent to one. We are only concerned here with the strong version; its role is to provide one answer to the question ‘What is a process?’; the answer is ‘A bisimilarity-class of process expressions’. This recalls the definition of a natural number, attributed to Bertrand Russell, as ‘an equivalence class of sets under bijection’.

CSP places emphasis on the failures preorder on process expressions; this is—intentionally—much weaker than strong bisimilarity, and as a preorder it represents the way in which a process may be said to satisfy a specification. A point relevant to the present work is that, since the failures preorder includes the strong bisimilarity equivalence, it can be understood as a relation on *processes*, i.e. bisimilarity classes of process expressions, rather than on the process expressions themselves. In other words, the CSP semantics can be factored into two; first we determine which expressions denote the same process, and second we determine whether one process *satisfies* or *implements* another considered as a specification.

The remainder of this paper therefore has two concerns: First, how does (strong) bisimilarity relate to structural congruence as we have defined it, namely as the transitive reflexive symmetric closure of unfolding? Second, which of the well-known equations of CSP—as for instance listed in Hoare [7]—already hold for bisimilarity, and which hold only for failures equivalence?

Our first result is that bisimilarity is no stronger than structural congruence:

Theorem 3 (bisimilarity includes structural congruence). $\equiv \subseteq \sim$.

Proof. It is enough to show that \equiv is a bisimulation, and this is straightforward. \square

The second result is more surprising, since we are used to thinking of structural congruence as a demanding (i.e. small) congruence. But the version of structural congruence introduced here places more emphasis upon similar *behaviour*, and less upon *syntactic* similarity. In fact:

Theorem 4 (structural and behavioural congruence). *In finite (i.e. recursion-free) CSP, structural congruence and strong bisimilarity coincide; that is, $\equiv = \sim$.*

Proof. It only remains to prove that $\sim \subseteq \equiv$. Unfolding in finite CSP is both confluent and strongly normalising, so each term unfolds uniquely to a normal form (i.e. containing no process constructions except for enriched choice). Hence it is enough to prove that if two normal forms are bisimilar, they are structurally congruent. This can be proved by induction on the structure of normal forms. \square

Why does this result fail in the presence of recursion? It is enough to find two recursively defined processes that are bisimilar but not structurally congruent. This is easy; consider the two recursive definitions

$$A \hookrightarrow \{x \mapsto \{A\}\} \text{ and } B \hookrightarrow \{x \mapsto \{B\}\} .$$

They are not structurally congruent, since they never unfold to the same process, but they are bisimilar; indeed, consider the singleton bisimulation $\{(A, B)\}$.

Since bisimilarity does not coincide with structural congruence in the presence of recursion, we have to prove it to be a congruence by other means. In fact we must prove that it is preserved by each of the operators. This, though a little tedious, is straightforward.

Theorem 5 (congruence of bisimilarity). *Strong bisimilarity is a congruence.*

Proof (outline). We shall be content to present the case that one operator, \parallel , preserves bisimilarity. The proof of congruence of the other operators follows the same template.

We wish to prove that if $P_1 \sim P_2$ then $P_1 \parallel Q \sim P_2 \parallel Q$. For this, it is enough to prove that $\mathcal{B} \stackrel{\text{def}}{=} \{(P_1 \parallel Q, P_2 \parallel Q) \mid P_1 \sim P_2\}$ is a bisimulation.

Let $P_1 \parallel Q \searrow_{\mu} R_1$. We must find R_2 such that $P_2 \parallel Q \searrow_{\mu} R_2$ and $(R_1, R_2) \in \mathcal{B}$. Now by definition of transition we have $P_1 \parallel Q \hookrightarrow^* H$, with $R_1 \in H(\mu)$.

We consider only the case in which $\mu = z \in \alpha P_1 \setminus \alpha Q$ (other cases are similar). By the definition of unfolding for \parallel , we have $P_1 \hookrightarrow^* F_1$ and $Q \hookrightarrow^* G$, with $H(z) = F_1(z) \parallel G$, and $R_1 = P'_1 \parallel G$ where $P'_1 \in F_1(z)$.

From this we first deduce that $P_1 \searrow_z P'_1$. But $P_1 \sim P_2$, so there exists P'_2 such that $P_2 \searrow_z P'_2$ and $P'_1 \sim P'_2$. This implies that $P_2 \hookrightarrow^* F_2$, with $P'_2 \in F_2(z)$.

It follows that $P_2 \parallel Q \hookrightarrow^* F_2 \parallel G$ and $F_2 \parallel G \hookrightarrow H'$ for some (unique) H' ; so, by the transition rule, $P_2 \parallel Q \searrow_z R'_2$ for any R'_2 in $H'(z)$. But $H'(z) = F_2(z) \parallel G$ in this case, and $P'_2 \parallel G$ is in $F_2(z) \parallel G$. Hence $P_2 \parallel Q \searrow_z P'_2 \parallel G$. So we are done by taking R_2 to be $P'_2 \parallel G$. \square

What are we to make of our ‘discovery’ that, for finite CSP, bisimilarity (\sim) agrees with what we have called structural congruence (\equiv)? We have to admit that there is no unique notion of structure—and hence of structural congruence—for processes. This amounts to saying that, given a syntax of process terms, like CSP or CCS or ACP [2], there is no unique congruence on its terms such that we all agree that its congruence classes are processes. For some people, process structure involves a causal relationship among actions; in that case a process in which an action x causes an action y differs from one in which x merely precedes y . For other people, process structure will involve duration of action or locality of agents, and so on.

Our definition of unfolding commits us to an abstract notion of process in which causality, timing and placing are all ignored; the only remaining structure is that of non-determinism, recording the possible transitions in any state, whether or not under control of the process’s environment. This structure is captured for finite CSP by the congruence based upon unfolding, and—as we have seen—strong bisimilarity equally captures it.

How does this generalise to infinite CSP, where recursive definitions enable infinite unfolding? An infinitely proceeding process, one that enable an infinite sequence

$$P_1 \searrow_{\mu_1} P_2 \searrow_{\mu_2} P_3 \searrow_{\mu_3} \cdots$$

of transitions, corresponds closely to an infinite chain

$$S_1 \ni S_2 \ni S_3 \ni \cdots$$

of inverse membership of sets, in the theory of non-well-founded sets. Peter Aczel, in his book [1], argues that equality of such sets coincides with bisimilarity, i.e. it is the largest symmetric relation \mathcal{R} such that

$$\text{if } S \mathcal{R} T \text{ and } S' \in S, \text{ then } S' \mathcal{R} T' \text{ for some } T' \in T .$$

The bisimilarity of process terms differs only in that transition is a little more complex than inverse membership; indeed the relation \ni is replaced by the compound relation $\hookrightarrow^* \ni$. Therefore, even for infinite CSP, we are justified in interpreting bisimilarity as a structural congruence.

6 Equational Theory

In this section we examine the equational laws of CSP as given in Hoare's book [7], for concurrency (\parallel), general choice (\square), hiding ($\setminus Z$), interleaving ($\parallel\parallel$), and non-deterministic 'or' (\sqcap). They all hold when equality is interpreted as failures equivalence; that is the intention in CSP. We indicate which of them hold also for strong bisimilarity (\sim); in fact, we tag with the symbol \div all those—a minority—which do *not* so hold. As is well-known, the discrepancy between these two interpretations is largely due to the τ -transitions, which matter more in strong bisimilarity than in the failures model. They are relevant to the failures model in a negative sense: a *failure* of P is a pair (t, S) , where $t \in \mathcal{X}^*$ and $S \subseteq \mathcal{X}$, such that P can perform t to reach a state in which no τ -transitions are possible and in which no action in S can be performed.

Each law that holds for strong bisimilarity, \sim , can be established by exhibiting a suitable bisimulation; we omit these to avoid boring the reader. As mentioned above, a law is tagged with \div if it does not hold for \sim , and in each case we sketch a concrete counter-example. Many of these laws involve the non-deterministic 'or' (\sqcap); we will discuss this a little further at the end of this section.

Of course it has long been known that such laws do not hold for \sim . The point of this section is to add detail to our claim that the theory of CSP can be factorised into two complementary parts: first, we quotient the class of CSP expressions by the congruence \sim , calling each congruence class a *process*, and we identify an algebraic theory of these processes; second, we assert further laws that hold for the failures preorder over processes as so defined.

For the purpose of this section, let us first define the well-known CSP processes STOP_X and RUN_X as hnfs:

<i>L1</i>	$P \parallel Q = Q \parallel P$	
<i>L2</i>	$P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$	
<i>L3A</i>	$P \parallel \text{STOP}_{\alpha P} = \text{STOP}_{\alpha P}$	\div
<i>L3B</i>	$P \parallel \text{RUN}_{\alpha P} = P$	
Let $a \in (\alpha P \setminus \alpha Q)$, $b \in (\alpha Q \setminus \alpha P)$, and $\{c, d\} \subseteq (\alpha P \cap \alpha Q)$.		
<i>L4A</i>	$(c \rightarrow P) \parallel (c \rightarrow Q) = c \rightarrow (P \parallel Q)$	
<i>L4B</i>	$(c \rightarrow P) \parallel (d \rightarrow Q) = \text{STOP}$ if $c \neq d$	
<i>L5A</i>	$(a \rightarrow P) \parallel (c \rightarrow Q) = a \rightarrow (P \parallel (c \rightarrow Q))$	
<i>L5B</i>	$(c \rightarrow P) \parallel (b \rightarrow Q) = b \rightarrow ((c \rightarrow P) \parallel Q)$	
<i>L6</i>	$(a \rightarrow P) \parallel (b \rightarrow Q) = (a \rightarrow (P \parallel (b \rightarrow Q))) \mid b \rightarrow ((a \rightarrow P) \parallel Q)$	
Let $P = (x : X \rightarrow P(x))$ and $Q = (y : Y \rightarrow Q(y))$.		
<i>L7</i>	$(P \parallel Q) = (z : Z \rightarrow P' \parallel Q')$	
	where $\begin{cases} Z = (X \cap Y) \cup (X \setminus \alpha Q) \cup (Y \setminus \alpha P) \\ P' = P(z) \text{ if } z \in X \text{ otherwise } P' = P \\ Q' = Q(z) \text{ if } z \in Y \text{ otherwise } Q' = Q. \end{cases}$	

Table 1 Equational laws for \parallel from §2.3.1

$$\begin{aligned} \text{STOP}_X &\triangleq \{\mu \mapsto \emptyset \mid \mu \in X \cup \{\tau\}\}, \\ \text{RUN}_X &\triangleq \{x \mapsto \{\text{RUN}_X\} \mid x \in X\} \cup \{\tau \mapsto \emptyset\}. \end{aligned}$$

As described in [7], STOP_X is the process with alphabet X which never actually engages in any of the events of X . On the other hand RUN_X is the process which at all times can engage in any event of its alphabet X . We shall discuss these definitions further at the end of the section.

In Table 1 we list the equational laws for \parallel as presented in §2.3.1 in [7]. We note that all the laws hold for \sim except for *L3A*. The reason for the latter is that if P can perform a τ -transition then $P \parallel \text{STOP}_{\alpha P}$ can perform one, whereas $\text{STOP}_{\alpha P}$ cannot.

In Table 2 we list the equational laws for the general choice \square from §3.3.1. For this operator several of the laws do not hold for \sim . For *L1*, the reason is that the process $P \square P$ can possibly perform two τ -transitions when P can only perform one.

As expected the laws relating \square and \sqcap do not hold for \sim . For *L5*, the left-hand side must choose either $P(a)$ or $Q(a)$ when performing a shared event a whereas this choice is “delayed” in the right-hand side. For *L6*, if $P \searrow_{\tau} P'$ then the left-hand side may become $P' \square (Q \sqcap R)$, while the right-hand side becomes either $P' \square Q$ or $P' \square R$. Similarly, we can find a counter-example to *L7* by letting Q perform a τ -transition.

Table 3 contains the laws governing the hiding operator. All of the laws hold for \sim , except for two. For *L5*, suppose that the choice x is hidden by Z , i.e. $x \in Z$. Then the left-hand side can perform a τ -transition to become $P \setminus Z$, but this τ -transition is not possible for the right-hand side of the equation. For *L9* we tacitly assume that we have generalised the binary non-deterministic ‘or’ operator into an n -ary operator. The law contravenes \sim for the same reason as *L5*.

Table 4 contains the laws for the interleaving operator. As expected the law *L1* does not hold for \sim . A simple counter-example can be constructed by making P perform any transition. For *L5*, $P \parallel \text{RUN}$ can possibly perform a τ -transition (if P can) which cannot be matched by RUN .

<i>L1</i>	$P \sqcap P = P$	\div
<i>L2</i>	$P \sqcap Q = Q \sqcap P$	
<i>L3</i>	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$	
<i>L4</i>	$P \sqcap \text{STOP} = P$	
<i>L5</i>	$(x : X \rightarrow P(x)) \sqcap (y : Y \rightarrow Q(y)) =$ $(z : (X \cup Y) \rightarrow \begin{cases} P(z) & \text{if } z \in (X \setminus Y) \\ Q(z) & \text{if } z \in (Y \setminus X) \\ P(z) \sqcap Q(z) & \text{if } z \in (X \cap Y) \end{cases})$	\div
<i>L6</i>	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap (P \sqcap R)$	\div
<i>L7</i>	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap (P \sqcap R)$	\div

Table 2 Equational laws for \sqcap from §3.3.1

<i>L1</i>	$P \setminus \{\} = P$	
<i>L2</i>	$(P \setminus Y) \setminus Z = P \setminus (Y \cup Z)$	
<i>L3</i>	$(P \sqcap Q) \setminus Z = (P \setminus Z) \sqcap (Q \setminus Z)$	
<i>L4</i>	$\text{STOP}_X \setminus Z = \text{STOP}_{X \setminus Z}$	
<i>L5</i>	$(x \rightarrow P) \setminus Z = \begin{cases} x \rightarrow (P \setminus Z) & \text{if } x \notin Z \\ P \setminus Z & \text{if } x \in Z \end{cases}$	\div
<i>L6</i>	If $\alpha P \cap \alpha Q \cap \alpha Z = \{\}$, then $(P \parallel Q) \setminus Z = (P \setminus Z) \parallel (Q \setminus Z)$	
<i>L8</i>	If $Y \cap Z = \{\}$, then $(x : Y \rightarrow P(x)) \setminus Z = (x : Y \rightarrow (P(x) \setminus Z))$	
<i>L9</i>	If $Y \subseteq Z$, and Y is finite and not empty, then $(x : Y \rightarrow P(x)) \setminus Z = \prod_{x \in Y} (P(x) \setminus Z)$	\div

Table 3 Equational laws for \setminus from §3.5.1

<i>L1</i>	$P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R)$	\div
<i>L2</i>	$P \parallel Q = Q \parallel P$	
<i>L3</i>	$P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$	
<i>L4</i>	$P \parallel \text{STOP} = P$	
<i>L5</i>	$P \parallel \text{RUN} = \text{RUN}$ Provided P does not diverge	\div
<i>L6</i>	$(x \rightarrow P) \parallel (y \rightarrow Q) = (x \rightarrow (P \parallel (y \rightarrow Q))) \sqcap (y \rightarrow ((x \rightarrow P) \parallel Q))$	
<i>L7</i>	If $P = (x : X \rightarrow P(x))$ and $Q = (y : Y \rightarrow Q(y))$ then $P \parallel Q = (x : X \rightarrow (P(x) \parallel Q)) \sqcap (y : Y \rightarrow (P \parallel Q(y)))$	

Table 4 Equational laws for \parallel from §3.6.1

Finally, Table 5 contains the laws for non-deterministic ‘or’. Due to the simple unfolding rule for \sqcap , we are in fact able to prove that the three first laws hold even for structural congruence (\equiv), by reducing both sides to the same hnf. However, laws *L4* and *L5* do not hold for \sim since the left-hand side has not made the choice between the processes, whereas the choice has been made in the right-hand side. For laws *L6* and *L7* we can find simple counter-examples by letting respectively P or R perform a transition. This will leave the non-deterministic ‘or’ unchanged in the left-hand side of the equation, but enforce a choice in the right-hand side.

We note that we cannot redefine \sqcap in such a way that these laws hold for \sim . That’s because the laws take advantage of the properties of the failures preorder, which intentionally conflates what may be regarded as two kinds of non-determinism: the

<i>L1</i>	$P \sqcap P = P$	
<i>L2</i>	$P \sqcap Q = Q \sqcap P$	
<i>L3</i>	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$	
<i>L4</i>	$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$	\div
<i>L5</i>	$(x : X \rightarrow (P(x) \sqcap Q(x))) = (x : X \rightarrow P(x)) \sqcap (x : X \rightarrow Q(x))$	\div
<i>L6</i>	$P \parallel (Q \sqcap R) = (P \parallel Q) \sqcap (P \parallel R)$	\div
<i>L7</i>	$(P \sqcap Q) \parallel R = (P \parallel R) \sqcap (Q \parallel R)$	\div

Table 5 Equational laws for \sqcap from §3.2.1

dynamic kind arising from uncontrolled decisions at run-time, and the static kind arising from lack of knowledge of what process is running. The latter kind is what lies behind the failures preorder. But bisimilarity aims to define the notion of non-deterministic process (irrespective of how much we know about it), so it is only concerned with the former kind.

We conjecture that further CSP operators—for example sequential composition and interrupt—are amenable to definition by unfolding, and that certain of the laws they satisfy will hold for \sim . Recall our conjecture that the constant SKIP_X , essential for sequential composition, can be treated as a special normal form, whose character is determined by unfolding rules. We also conjecture that both STOP_X and RUN_X , which we defined above, can instead be treated as special normal form constants, in such a way that the laws listed for them will all hold for \sim .

7 Conclusion

This paper is motivated by the wish to represent CSP as a bigraphical reactive system. This is to ensure that the bigraph model, which aspires to be a generic framework for process modelling, can indeed represent the phenomena that are special to CSP. This will not only add validity to the bigraph model; it will also allow other bigraphical reactive systems to benefit from the specific insights of CSP.

In this paper we have therefore handled some of CSP’s repertoire of operators in a way that reflects how they will be defined in bigraphs, while remaining close to CSP’s syntactic framework. In bigraphs, we expect to define these operators by generalising a notion of unfolding already present in bigraphs (see [11]). This generalisation has already been mooted, and the CSP encoding will add motivation for it.

Indeed, the work presented here suggests a class of unfolding relations for which unfolding in bigraphs is guaranteed to be confluent, though it is not yet clear how broad this class may be. After doing the present work the authors have become aware of Roscoe’s [16] proposed definition for ‘CSP-like’ operators; it will be interesting to compare CSP-like operators with those that unfold confluent.

The second CSP phenomenon which we have mentioned, the ownership of channels, has not been discussed further in this paper. It will be handled in bigraphs by

imposing a constraint on what bigraphs are admitted in the encoding. As mentioned in the introduction, such a constraint in bigraphs is called a *sorting*; a sorting is usually needed when encoding a calculus, because of the generality of bigraphical contexts.

Let us renew our claim that the present approach complements, rather than contravenes, the original presentation of CSP. It enriches the choice construction, making it non-deterministic. Hoare's original choice construction preserves determinism, thus allowing the phenomena of concurrency to be introduced one-by-one as the repertoire of operators is extended. But the original choice construction is still available as a special case of the enriched one. Also, our hnfs provide a simple formal understanding of many of CSP's wide range of operators, and a simple way to define the bisimilarity of CSP process expressions. As we have noted, most of CSP's equational properties, not involving \square , hold not just for the failures preorder, but even for strong bisimilarity.

In doing this work we have gained a more intimate understanding of the power and beauty of CSP, such as cannot be gained by just reading about it. We would be grateful for any feedback from those who have worked closely with CSP, in the hope of a better integrated theory of concurrent processes.

References

1. P Aczel, *Non-well-founded Sets*, CSLI Lecture Notes 14, Center for the Study of Language and Information (1988).
2. J Baeten and W P Weijland, *Process Algebra*, Cambridge University Press (1990).
3. S Brookes, A W Roscoe and D J Walker, An operational semantics for CSP. (Manuscript). Available from <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/brookes/www/papers/OperationalSemanticsCSP.pdf> (1988).
4. N Dershowitz, Z Manna, *Proving termination with multiset orderings*. Communications of the ACM, Vol 22, Issue 8, pages 465–476, ACM (1979).
5. R De Nicola, *Two complete axiom systems for a theory of Communicating Sequential Processes*. Information and Control, Vol 64, pages 136–172 (1985).
6. C A R Hoare, *Communicating Sequential Processes*. Communications of the ACM, Vol 21, Issue 8, pages 666–777, ACM (1978).
7. C A R Hoare, *Communicating Sequential Processes*. Prentice Hall (1985).
8. R Milner, *Synthesis of Communicating Behaviour*. Proc. 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol 64, Springer-Verlag (1978).
9. R Milner, *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, Vol 92, Springer-Verlag (1980).
10. R Milner, *Communicating and Mobile Systems: the π -Calculus*. Cambridge University Press (1999).
11. R Milner, *The Space and Motion of Communicating Agents*. Cambridge University Press (2009).
12. D Park, *Concurrency and Automata on Infinite Sequences*. Lecture Notes in Computer Science, Vol 104, Springer-Verlag (1980).
13. G D Plotkin, A structural approach to operational semantics. Report DAIMI-FN-19, Computer Science Department, Århus University, Denmark (1981). Reprinted as [14].

14. G D Plotkin, *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming, Vol 60–61, pages 17–139, Elsevier (2004).
15. A W Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall (1998); revised (2005).
16. A W Roscoe, *On the expressiveness of CSP*, forthcoming paper (2009).
17. TeReSe, Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science 55, Cambridge University Press (2003).