Possible and Guaranteed Concurrency in CSP

Marta Kwiatkowska*
School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

Iain Phillips†
Department of Computing, Imperial College
180 Queens Gate, London SW7 2BZ, UK

Abstract

As part of an effort to give a "truly concurrent" semantics to process algebra, we propose a framework of refinements of the failures model for CSP with concurrency, conflict and causality relations on traces. These relations are defined by induction over syntax of CSP processes. We study in detail two new semantics: the *possible concurrency* (where two traces are said to be concurrent if they *may* be observations of the same concurrent run) and the *possible conflict* (two traces are said to be in conflict if they *may* be observations of two different runs). The *guaranteed concurrency* is obtained from the possible conflict semantics. Although the expansion law is necessarily weakened to an inequality, we show that most of the CSP laws are preserved, the exception being the idempotency of choice for the possible conflict refinement. Finally, we show that our semantics is well-founded by demonstrating a strong connection with the existing event structures semantics for CSP. The latter results show that, in a certain sense, concurrency distinctions can be made at the level of syntax, without resorting to reasoning about event occurrences.

1 Introduction

This paper is part of an effort by many authors to give a "truly concurrent" semantics to process algebra. Many authors have previously contributed in this area, see *e.g.* [22, 19, 9, 20, 4, 11, 15, 1, 17, 2], and the methods have included translation into Petri nets, deriving an event structure, and enhancing the labels in labelled transition systems, often leading to descriptions of low-level character. We propose starting at the high level of abstraction usually associated with process algebras, and define a refinement of the existing failures model directly from the syntax of CSP terms. The virtues of abstraction have long been recognised in software engineering and specification, and this is where this approach may be utilised.

As the touchstone for whether a semantics is truly concurrent, as usual we take the example processes $a \parallel b$ and $a{\cdot}b + b{\cdot}a$, where $\parallel$ denotes parallel

*This research was started when the author was Visiting Academic at the Department of Computing, Imperial College supported by the Nuffield Science Foundation grant (SCI/124/528/G).

†Supported by SERC grant GR/F72475.

composition, $\cdot$ denotes sequential composition and $+$ denotes choice. These two are of course equated in the standard interleaving semantics of process algebras such as CCS, CSP, ACP due to the expansion law. However, we would expect them to be distinguished by a truly concurrent semantics, since $a \parallel b$ exhibits concurrency (in that actions a and b are causally independent), whereas $a{\cdot}b + b{\cdot}a$ does not.

Trace theory [16] provides one solution to the example. The sequences of possible actions are just ε, a, b, ab, ba in each case. In the case of $a \parallel b$, since a, b are independent, denoted $a \ \iota \ b$, we say that ab and ba are equivalent, since they are the same except that independent actions are permuted. However, in the case of $a{\cdot}b + b{\cdot}a$ the sequences ab, ba are not equivalent, and so we have distinguished the processes.

The disadvantage of trace theory is that it does not handle non-determinism and deadlock. For instance the processes $a{\cdot}b$ and $a + a{\cdot}b$ have the same sequences, namely ε, a, ab, and there is no independence between actions in either case. However, the second process may deadlock after performing a, so that a distinction should be made. It is for this reason that Milner was led to consider a semantics for processes more refined than that used in formal language theory. He proposed observation equivalence (bisimulation), which takes us a long way from traces, and is generally considered the most refined behavioural equivalence. Brookes, Hoare and Roscoe [5] took a different approach, in which they refined the traces[1] model with deadlock information telling us which actions may be refused after some sequence of actions, giving the so-called *failures* model.

We aim to obtain a semantics for CSP-like languages capable of making concurrency distinctions as in Mazurkiewicz trace theory, while, at the same time, capable of handling non-determinism and deadlock in the process algebra style. An extension of trace theory to cater for Petri nets appeared in [13]. We propose a new framework of *refinements* of the classical, failures semantics with relations on traces capturing the notions of *possible*, and their duals, *guaranteed*, relations of concurrency, conflict and causality. These relations are defined by induction over syntax of CSP processes. We discuss two refinements of the failures semantics (with possible concurrency and conflict) and the corresponding laws of the calculi. Finally, we demonstrate a strong connection with the event structures semantics for CSP.

This paper had its origins in a preliminary report presented at an internal workshop [14], where the idea of a conflict relation on traces was introduced. In this paper, we propose a whole new framework as outlined above, including material from [14] for the sake of completeness. A true concurrency semantics for CSP, without the inequational version of the expansion law, appeared in [20]. It would be interesting to compare the two approaches.

The paper is organised as follows. Section 2 introduces the central idea, while Section 3 contains an overview of CSP. In Section 4 we define two main

[1] Note that in the CSP world "trace" means a sequence of actions performed by a process, whereas for trace theory, it means an equivalence class of such sequences under the equivalence generated by the independence relation.

refinements of the failures model and summarise the corresponding laws. Finally, Section 5 states a formal connection of our abstract semantics with event structures.

2 Discussion: "may" or "must" concurrency?

The equational version of the expansion law is a natural consequence of the statement that concurrency is not observable. As a result, concurrency is only structural, and not behavioural, since the expansion law acts as a "rewrite" rule on syntactic forms. This also means that syntactic distinctions are ignored at the behavioural level. We do not argue with the view that concurrency is not observable. Instead, we propose to view non-interleaving theories as based on an element of knowledge about the internal structure of the system. One may say that the issue is approached from a different perspective, that of a *system designer*, rather than a *system observer*.

The aim of this paper is to obtain a semantic model for CSP-like languages which

- is *non-interleaving*, in the sense that it does not satisfy the expansion law, but instead it satisfies its weaker, inequational version;
- is *simple* and *abstract*, *i.e.* makes sense of processes such as $a \,||\, b + (a{\cdot}b)$ without resorting to relabelling;
- perturbs *classical semantics* as little as possible, *i.e.* it preserves as many laws as possible and does not reject classical semantics in favour of existing non-interleaving models.

Much has been published on the subject of non-interleaving semantics for process algebras. The advantage of models such as Petri nets, event structures and pomsets is that they are concrete, intuitive and graphical, and can therefore serve as design tools. However, they can be difficult to reason about, and not sufficiently abstract. Often two levels are introduced, one for event occurrences and one for abstract actions. This is cumbersome to use, as reasoning about a system must involve both levels. Furthermore, there is too much dependence on how particular nets or event structures are manufactured, and too much arbitrariness in the way the maps on processes are defined, possibly leading to inconsistencies with operational intuitions.

We feel that, alongside models such as event structures, there is much to gain from complementing existing interleaving theories with abstract, non-interleaving ones. As we are hoping to show in this paper, non-interleaving *can* be handled at a high level of abstraction, and can, therefore, support *meaningful specifications*. The motivation for using non-interleaving is that it adequately represents phenomena present in hardware (*e.g.* at the logic gate level), and also leads to more efficient verification of certain classes of properties (see *e.g.* [10]).

The central idea of this paper is to *refine* the existing classical semantics, *e.g.* failures in the case of CSP, with additional concurrency information. Our intention is that this is defined by induction over syntax of process expressions, in the same way as the failures themselves can be defined directly from syntax (see Section 4.1). The idea is as follows. Consider a process algebra language. We think of traces of a given process P as *observations*. We propose to refine the existing semantics with a binary *concurrency* relation (denoted *conc*) on traces meaning two traces are in the relation iff they are observations of the same concurrent run. For example, we would have ab *conc* ba in $a \parallel b$, but not in $ab + ba$. Note that this is more general than the approach used, say, in Mazurkiewicz trace theory, where the concurrency relation is on actions, because it can handle context dependency. However, there is a difficulty with non-determinism. Consider the process:

$$(a \parallel b) + (a \cdot b + b \cdot a)$$

Clearly, in the subprocess on the left ab *conc* ba, while in the subprocess on the right this is *not* the case. Should ab *conc* ba hold or not in *the whole* process? One could rename or index the actions to distinguish between "the a on the left" and "the a on the right", but this introduces two levels of reasoning, which we believe is unnecessary. Instead we propose to make a choice between interpreting the concurrency relation either as:

> it is *possible* for ab to be concurrent with ba, in the sense that *there exists* a concurrent run such that both traces can be observed from it;

or

> it is *guaranteed* that ab is concurrent with ba, in the sense that *for every* concurrent run both traces can be observed.

We shall refer to the first interpretation above as the *possible* (also "*may*") concurrency (denoted $co()$), while the latter will be called *guaranteed* (also "*must*") concurrency and will be denoted $gco()$.

Formally, given a process expression P, we shall distinguish an indexed family of relations

$$co(P) \subseteq tr(P)^2$$

defined by induction over syntax, where $tr(P)$ denotes the set of traces of P.

We would anticipate that the two interpretations above will correspond to different sets of laws. The decision regarding which interpretation to use would depend on application; we feel that guaranteed concurrency would correspond to measuring the amount of concurrency, while possible concurrency to describing low-level behavioural phenomena. This can be explained as follows. In the guaranteed concurrency semantics we obtain $a \cdot b + b \cdot a \leq a \parallel b$, which can be understood as "the more concurrent the process, the higher it is in the ordering". On the other hand, the possible concurrency semantics yields $a \parallel b \leq a \cdot b + b \cdot a$: the process lower in the ordering has more complex behaviour (a, b can be seen as either ordered in time or in indeterminate order) than the process higher in the ordering. Thus, "the more abstract (less complex) the process, the higher it is in the ordering".

We also note that, independently of deriving the (possible or guaranteed) concurrency, one may also consider relations on traces corresponding to the (possible or guaranteed) *conflict* or *causality* relations. Roughly speaking, two traces would be in a *possible conflict* relation if they *may* be observed of two different runs, originating from either side of the choice operator. The *guaranteed conflict* would be the obvious dual. The definition of the causality relations would be similar, and would correspond to the sequential composition of processes.

These relations can be combined with the classical semantics to give models of varying powers of distinction. Concurrency and conflict will also be related under complement – some approximation of, say, guaranteed concurrency can be obtained from possible conflict, as we shall show later.

In this paper, we shall define refinements of the failures semantics for CSP with possible concurrency and possible conflict relations. We believe, however, that the above ideas are applicable to a range of process algebras, although we should emphasise that the particular definitions and results presented here are dependent on the semantics of the CSP mixed parallel. In the full paper we shall also consider other CSP parallel operators known in the literature.

As proposed in general observation structures [8], other notions of observations, *e.g.* failures, may also be used to replace traces, but we should point out that using traces has obvious advantages for complexity of specification and verification.

3 Preliminaries

3.1 Notation

We shall use the following notation conventions throughout the paper. Act is a set of action symbols[2]. The set of finite strings (also called *traces*) over the set Act is denoted by $Act^\star$; ε is the empty trace. We use a, b, c to range over Act, and s, t, u to range over $Act^\star$. Given $a \in Act$ and traces $s, t \in Act^\star$, we write as, sa and st for the respective concatenations. Given a set L, the set of all finite subsets of L will be denoted by $\mathcal{P}_f(L)$, and the set of all its subsets by $\mathcal{P}(L)$. We use A, B, X, Y to range over $\mathcal{P}_f(Act)$. $s \backslash d$, where $d \in Act$, denotes *hiding*, *i.e.* the string s with all occurrences of d deleted. Given $A \in \mathcal{P}_f(Act)$, $s \lceil A$ denotes *restriction*, *i.e.* the string s with all occurrences of actions outside the set A deleted.

3.2 The language of CSP

We use a subset of Communicating Sequential Processes (CSP) as described in [6]. This is essentially an alphabet-free version of CSP as introduced in [12]. In this paper only a summary of the language and the model is presented. We refer the reader to [12, 5, 6] for more details.

[2]We prefer to use the term "action" instead of "event" so as to distinguish between actions and their occurrences; the latter are usually referred to as "events".

The syntax of CSP process expressions is as follows:

$$P ::= 0 \mid a{\cdot}P \mid P \sqcap Q \mid P \,\Box\, Q \mid P_A \parallel Q_B \mid P \backslash a \mid f(P) \mid x \mid \mu x.P$$

where P, Q ranges over (syntactic) process expressions, a is an action ranging over the set Act, A, B are (finite) sets of actions, and x is a process variable. 0 represents a deadlocked process (usually denoted STOP). $a{\cdot}P$ denotes prefixing the process P with the action a (usually $a \rightarrow P$). We shall often elide $\cdot$ and 0; *e.g.* the process $a{\cdot}b{\cdot}0$ will simply be denoted by ab. $P \sqcap Q$ is the non-deterministic choice (internal), while $P \,\Box\, Q$ the deterministic choice (external). $P_A \parallel P_B$ is the mixed parallel operator; here A is the named alphabet for the process P, B the named alphabet for the process Q, and $P_A \parallel Q_B$ requires simultaneous participation of P and Q on any actions belonging to $A \cap B$, but the process P may progress independently on actions belonging to $A - B$ (and symmetrically for Q). This operator is a slight generalisation of the parallel operator introduced by Hoare in [12], where alphabets of processes were used in place of arbitrary sets A and B. If it is clear from the context, the alphabets will be omitted. $P \backslash a$ denotes the hiding operator, while $f(P)$ renaming. We assume that $f : Act \longrightarrow Act$ is a 1-1 function. This restriction guarantees no auto-concurrency, and is convenient when considering the relationship with event structures. Finally, recursion is handled by terms $\mu x.P$, where μ is a variable binding operator.

We do not consider the pure interleaving parallel $|||$ and the sequential composition ; of processes. We anticipate that the framework can be extended straightforwardly to cater for sequential composition. The pure interleaving parallel can yield autoconcurrency.

We shall denote the set of terms defined by the above syntax by CSP.

4 Semantics

In this section we define the *possible conflict* and *possible concurrency* semantics for CSP. We begin by reviewing the failures semantics. We also study the notion of *guaranteed concurrency* via the notion of possible conflict, in a sense that will be made precise in Section 5.

4.1 Failures semantics

The failures semantic model [5] views a process as a set of *failures*, that is, pairs (s, X) where $s \in Act^\star$ is a trace which the process *may* perform and X is a finite set of actions which the process *may* refuse after having done s. Formally, a failure set is a subset:

$$F \subseteq Act^\star \times \mathcal{P}_f(Act)$$

satisfying the following four conditions:

Table 1: The failures table. $\mathrm{C}(X, Y, Z)$ denotes $X \subseteq A, Y \subseteq B, Z \cap (A \cup B) = \emptyset$.

$$\frac{}{(\varepsilon, \emptyset) \in \mathit{fail}(0)} \qquad \frac{a \notin X}{(\varepsilon, X) \in \mathit{fail}(aP)}$$

$$\frac{(s, X) \in \mathit{fail}(P)}{(as, X) \in \mathit{fail}(aP)} \qquad \frac{(s, X) \in \mathit{fail}(P)}{(s, X) \in \mathit{fail}(P \sqcap Q)}$$

$$\frac{(s, X) \in \mathit{fail}(P)}{(s, X) \in \mathit{fail}(Q \sqcap P)} \qquad \frac{(s, X) \in \mathit{fail}(P)}{(s, X) \in \mathit{fail}(P \,\Box\, Q)}\; s \neq \varepsilon$$

$$\frac{(s, X) \in \mathit{fail}(P)}{(s, X) \in \mathit{fail}(Q \,\Box\, P)}\; s \neq \varepsilon \qquad \frac{(\varepsilon, X) \in \mathit{fail}(P) \quad (\varepsilon, X) \in \mathit{fail}(Q)}{(\varepsilon, X) \in \mathit{fail}(P \,\Box\, Q)}$$

$$\frac{(s, X \cup \{a\}) \in \mathit{fail}(P)}{(s \backslash a, X) \in \mathit{fail}(P \backslash a)} \qquad \frac{(s, X) \in \mathit{fail}(P)}{(f(s), f(X)) \in \mathit{fail}(f(P))}$$

$$\frac{(s \restriction A, X) \in \mathit{fail}(P) \quad (s \restriction B, Y) \in \mathit{fail}(Q) \quad \mathrm{C}(X, Y, Z)}{(s, X \cup Y \cup Z) \in \mathit{fail}(P_A \parallel Q_B)}\; s \in (A \cup B)^\star$$

(F1) $(\varepsilon, \emptyset) \in F$;
(F2) $(st, \emptyset) \in F \Rightarrow (s, \emptyset) \in F$;
(F3) $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$;
(F4) $(s, X) \in F \wedge (sc, \emptyset) \notin F \Rightarrow (s, X \cup \{c\}) \in F$.

For any set F of failures we define

$$tr(F) \quad = \quad \{s \mid \exists X.(s, X) \in F\}.$$

The set of failures is denoted by $\mathbb{F}$ and the ordering is as follows. We say $F \sqsubseteq F'$ (F is more non-deterministic than F') iff $F \supseteq F'$.

There exists a refinement of the failures model with divergences [6] needed to ensure that the law $P \parallel \mathrm{CHAOS} = \mathrm{CHAOS}$ is satisfied. For reasons of space we shall not consider divergences in this paper, but an extension to include them is straightforward.

CSP can be given denotational semantics by means of a compositional mapping $\mathit{fail}(\cdot)$ from process expressions to the failure sets [6] summarised in Table 1. The treatment of recursion is standard and depends on the operators being monotone and continuous in the domain $(\mathbb{F}, \sqsubseteq)$.

4.2 The refined semantics

We shall add an extra component to the model, in the shape of a binary relation C on $tr(F)$. This will be either the possible concurrency relation or the possible conflict relation, which we define below.

Let $\mathbb{FC} = \{\langle F, C\rangle \mid F$ is a failure set, C a binary relation on $tr(F)\}$. An ordering is defined on $\mathbb{FC}$ as follows:

$$\langle F, C\rangle \sqsubseteq \langle F', C'\rangle \quad \iff \quad F \supseteq F' \text{ and } C \supseteq C'.$$

This is clearly a refinement of the usual failures ordering.

We may think of the pairs $\langle F, C\rangle$ as the *possibilities* of a process. Going up in the ordering corresponds to discarding possibilities.

As far as using $\mathbb{FC}$ as a semantic domain is concerned, the following will be enough:

$(\mathbb{FC}, \sqsubseteq)$ is a dcpo, with directed joins given by set intersection.

In fact, of course, we can say more, but our intention is to place further restrictions on the relations C to refine our model in the form of axioms similar to (F1)-(F4) above. Some ideas on how this may be done emerge from Section 5, where connections with event structures are explored.

We are going to define two alternative semantic maps from CSP terms to the domain $\mathbb{FC}$. The first corresponds to possible concurrency, and the second to possible conflict. We give them component-wise, so that they are

$$P \mapsto \langle \mathit{fail}(P), \mathit{co}(P)\rangle$$
$$P \mapsto \langle \mathit{fail}(P), \mathit{cf}(P)\rangle$$

The maps appear in Tables 1, 2 and 3.

As explained in Section 2, two traces s, t of a process P are possibly concurrent ($s\ \mathit{co}(P)\ t$) if they may be observations of the same run of P. They possibly conflict ($s\ \mathit{cf}(P)\ t$) if they may be observations of two conflicting runs of P. The style of our tables is to define $\mathit{co}(P)$ and $\mathit{cf}(P)$ by induction on syntax. Recall that we have $\mathit{co}(P), \mathit{cf}(P) \subseteq tr(P)^2$ as background conditions. The rules (sym) and (refl) are used for convenience, to avoid complicating the definition.

The crucial rule for $\mathit{co}()$ is (L-R-$\|$), which introduces concurrency. Any valid trace is concurrent with itself by the rule (refl). The other rules preserve it through the syntactic structure. As far as $\mathit{cf}()$ is concerned, the crucial rules are (L-R-$\Box$) and (L-R-$\sqcap$), which introduce conflict. Again, the other rules preserve it through the syntactic structure.

Example 4.1 Consider $a \parallel b$. $ab\lceil\{a\}\ \mathit{co}(a)\ ba\lceil\{a\}$ and $ab\lceil\{b\}\ \mathit{co}(b)\ ba\lceil\{b\}$ by rule (refl), and so $ab\ \mathit{co}(a \parallel b)\ ba$.

The main difference between the rules for internal and external choice is that, by (L-R-$\sqcap$), all traces of P, including the empty trace ε, are in conflict with each trace of Q. On the other hand, according to (L-R-$\Box$) only those pairs of traces from P and Q are in conflict which are not simultaneously empty. The rules can be illustrated graphically in terms of the following event structures. The process $\tau a \,\Box\, \tau b$ is depicted by:

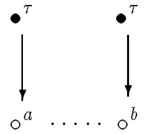


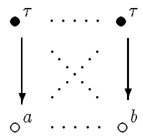
Table 2: The possible concurrency relation

(refl)	$\dfrac{s \in tr(P)}{s \;\; co(P) \;\; s}$	(sym)	$\dfrac{s \;\; co(P) \;\; s'}{s' \;\; co(P) \;\; s}$
(pref)	$\dfrac{s \;\; co(P) \;\; s'}{as \;\; co(a{\cdot}P) \;\; as'}$	(hide)	$\dfrac{s \;\; co(P) \;\; s'}{s\backslash a \;\; co(P\backslash a) \;\; s'\backslash a}$
(ren)	$\dfrac{s \;\; co(P) \;\; s'}{f(s) \;\; co(f(P)) \;\; f(s')}$	(L-R-$\parallel$)	$\dfrac{s\restriction A \;\; co(P) \;\; s'\restriction A \qquad s\restriction B \;\; co(Q) \;\; s'\restriction B}{s \;\; co(P_A \parallel Q_B) \;\; s'}$
(L-$\sqcap$)	$\dfrac{s \;\; co(P) \;\; s'}{s \;\; co(P \sqcap Q) \;\; s'}$	(R-$\sqcap$)	$\dfrac{s \;\; co(Q) \;\; s'}{s \;\; co(P \sqcap Q) \;\; s'}$
(L-$\Box$)	$\dfrac{s \;\; co(P) \;\; s'}{s \;\; co(P \Box Q) \;\; s'}$	(R-$\Box$)	$\dfrac{s \;\; co(Q) \;\; s'}{s \;\; co(P \Box Q) \;\; s'}$

Table 3: The possible conflict relation

(pref)	$\dfrac{s \;\; cf(P) \;\; s'}{as \;\; cf(a{\cdot}P) \;\; as'}$	(sym)	$\dfrac{s \;\; cf(P) \;\; s'}{s' \;\; cf(P) \;\; s}$
(hide)	$\dfrac{s \;\; cf(P) \;\; s'}{s\backslash a \;\; cf(P\backslash a) \;\; s'\backslash a}$	(ren)	$\dfrac{s \;\; cf(P) \;\; s'}{f(s) \;\; cf(f(P)) \;\; f(s')}$
(L-$\parallel$)	$\dfrac{s\restriction A \;\; cf(P) \;\; s'\restriction A}{s \;\; cf(P_A \parallel Q_B) \;\; s'}$	(R-$\parallel$)	$\dfrac{s\restriction B \;\; cf(Q) \;\; s'\restriction B}{s \;\; cf(P_A \parallel Q_B) \;\; s'}$
(L-$\sqcap$)	$\dfrac{s \;\; cf(P) \;\; s'}{s \;\; cf(P \sqcap Q) \;\; s'}$	(R-$\sqcap$)	$\dfrac{s \;\; cf(Q) \;\; s'}{s \;\; cf(P \sqcap Q) \;\; s'}$
(L-R-$\sqcap$)	$\dfrac{s \in tr(P) \;\; s' \in tr(Q)}{s \;\; cf(P \sqcap Q) \;\; s'}$		
(L-$\Box$)	$\dfrac{s \;\; cf(P) \;\; s'}{s \;\; cf(P \Box Q) \;\; s'}$	(R-$\Box$)	$\dfrac{s \;\; cf(Q) \;\; s'}{s \;\; cf(P \Box Q) \;\; s'}$
(L-R-$\Box$)	$\dfrac{s \in tr(P) \;\; s' \in tr(Q)}{s \;\; cf(P \Box Q) \;\; s'} \; s, s' \neq \varepsilon$		

while the process $a \sqcap b$ by:



(Here τ denotes hidden events and is used for purposes of illustration, arrows represent causality and conflict is denoted by dotted lines.) Note that in the event structure for $a \sqcap b$ the conflict arises for the hidden events and is inherited further on.

Observe that it is possible for a trace to be *self-conflicting*, *e.g.* $\varepsilon\ cf(a \sqcap b)\ \varepsilon$. Self-conflict corresponds to non-determinism.

Proposition 4.2 With both the $co()$ and $cf()$ semantics, all the operations on $\mathbb{FC}$ are monotone and continuous.

The proposition allows recursive processes to be handled by taking least fixed points in the standard way.

We shall show in Section 5 that the intuitive explanations of our semantic maps we have given in terms of runs can be justified in terms of event structures.

4.3 The laws

We have given two alternative semantics for the CSP operators. We now investigate to what extent the usual laws of CSP are preserved. The interleaving law $a \parallel b = ab \,\Box\, ba$ is where the differences appear most sharply. Of course, $fail(a \parallel b) = fail(ab \,\Box\, ba)$. However, $co(a \parallel b) \supseteq co(ab \,\Box\, ba)$, so that under the possible concurrency ($co()$) semantics

$$a \parallel b \quad \sqsubseteq \quad ab \,\Box\, ba.$$

On the other hand, $cf(a \| b) \subseteq cf(ab \,\Box\, ba)$, so that under the possible conflict $cf()$ semantics

$$a \parallel b \quad \sqsupseteq \quad ab \parallel ba.$$

Tables 4 and 5 show the laws which we have verified as sound for the $co()$ semantics. Tables 4 and 6 summarize the laws for the $cf()$ semantics. Notice that the idempotency of choice fails. Clearly $cf(P \sqcap P) = tr(P)^2$, so that $P \sqcap P$ has maximal conflict.

5 Connection with event structures

We adopt the modelling of CSP in prime event structures described in [15]. Let $\mathcal{E}(P)$ denote the event structure associated with CSP process P. We shall describe how we can give a concurrency / conflict semantics to event structures in a simple and intuitive way. Let $co(E)$, resp. $cf(E)$, denote the possible

Table 4: The laws of CSP with possible conflict and possible concurrency

$$
\begin{array}{rcll}
P \mathbin{\Box} Q & = & Q \mathbin{\Box} P & \\
P \mathbin{\Box} (Q \mathbin{\Box} R) & = & (P \mathbin{\Box} Q) \mathbin{\Box} R & \\
P \mathbin{\Box} 0 & = & P & \\
a(P \sqcap Q) & = & aP \mathbin{\Box} aQ & \\
P \sqcap Q & \sqsubseteq & P & \\
P \sqcap Q & = & Q \sqcap P & \\
P \sqcap (Q \sqcap R) & = & (P \sqcap Q) \sqcap R & \\
P \sqcap Q & \sqsubseteq & P \mathbin{\Box} Q & \\
P_A \parallel Q_B & = & Q_B \parallel P_A & \\
P_A \parallel (Q_B \parallel R_C)_{B \cup C} & = & (P_A \parallel Q_B)_{A \cup B} \parallel R_C & \\
P_A \parallel 0_B & = & P_A & \text{if } A \cap B = \emptyset \\
P_A \parallel 0_A & = & 0 & \\
(aP) \backslash a & = & P \backslash a & \\
(aP) \backslash b & = & a(P \backslash b) & \text{if } a \neq b \\
(P \sqcap Q) \backslash a & = & (P \backslash a) \sqcap (Q \backslash a) &
\end{array}
$$

Table 5: Additional laws of CSP with possible concurrency

$$
\begin{array}{rcll}
P \mathbin{\Box} P & = & P & \\
P \sqcap P & = & P & \\
P \mathbin{\Box} (Q \sqcap R) & = & (P \mathbin{\Box} Q) \sqcap (P \mathbin{\Box} R) & \\
aP_A \parallel bQ_B & \sqsubseteq & a(P_A \parallel bQ_B) \mathbin{\Box} b(aP_A \parallel Q_B) & \text{where } A \cap B = \emptyset, \\
 & & & a \in A, b \in B
\end{array}
$$

concurrency, resp. conflict, semantics of an event structure E. Then we shall establish

$$
\begin{array}{lcl}
s \; co(P) \; t & \text{iff} & s \; co(\mathcal{E}(P)) \; t \\
s \; cf(P) \; t & \text{iff} & s \; cf(\mathcal{E}(P)) \; t
\end{array}
$$

Table 6: Additional laws of CSP with possible conflict

$$
\begin{array}{rcll}
P \mathbin{\Box} P & \sqsubseteq & P & \\
P \mathbin{\Box} (Q \sqcap R) & \sqsupseteq & (P \mathbin{\Box} Q) \sqcap (P \mathbin{\Box} R) & \\
P \sqcap P \sqcap P & = & P \sqcap P & \\
aP_A \parallel bQ_B & \sqsupseteq & a(P_A \parallel bQ_B) \mathbin{\Box} b(aP_A \parallel Q_B) & \text{where } A \cap B = \emptyset, \\
 & & & a \in A, b \in B
\end{array}
$$

As well as providing evidence that our two semantics are well-founded, this will enable us to study the properties of our semantic domains more easily.

We start by defining the event structures which we shall be using. A *partially labelled prime event structure* $(E, \leq, \#, \ell)$ is a set E of events equipped with a partial order $\leq$ (causation), an irreflexive symmetric conflict relation $\#$, and a partial labelling function $\ell : E \longrightarrow Act$, satisfying

1. (axiom of finite causes) for every $e \in E, \{e' \mid e' \leq e\}$ is finite;
2. (forward propagation of conflict) if $e \# e' \leq e''$ then $e \# e''$.

Here Act is the set of actions which we earlier adopted for CSP. The labelling is partial because hidden events will be unlabelled. We shall refer to partially labelled prime event structure simply as event structures. We let $E, \ldots$ range over event structures, and $e, \ldots$ over events (members of E). Prime event structures were defined in [18].

We say that events e, e' of E are *concurrent* (notation $e\ co\ e'$) if none of $e \leq e'$, $e' \leq e$, $e \# e'$ holds. E is said to be *non-autoconcurrent* (nac) if concurrent events cannot have the same label, i.e., if $e, e' \in E$ are concurrent, then it is not the case that $\ell(e)$, $\ell(e')$ are both defined and equal to each other.

A *run* (or configuration) r of an event structure E is a conflict-free, downwards-closed subset of E. In other words:

1. if $e, e' \in r$ then not $e \# e'$;
2. if $e \in r$ and $e' < e$ then $e' \in r$.

We say that runs r, r' of E are *compatible* (denoted $r\ cp\ r'$) if $r \cup r'$ is a run. This amounts to seeing whether $r \cup r'$ is conflict-free, since downwards closure is automatic. If $r \cup r'$ is not a run then we say that r and r' *conflict* (denoted $r \# r'$).

Now we define what it means for a trace t to be an *observation* of a run r of an event structure E. A trace is a member of $Act^{\star}$. It is useful to regard such strings as also being linearly ordered multisets, in the obvious way. Let $obs_E(t, r)$ iff there is a partial bijection $f : r \rightharpoonup t$ such that

1. if $e \in r$ and $\ell(e)\downarrow$ then $f(e)\downarrow$ and $f(e) = \ell(e)$;
2. if $e, e' \in r$, $e \leq e'$ and $\ell(e), \ell(e')\downarrow$, then $f(e) \leq f(e')$

where $g(e)\downarrow$ means $g(e)$ is defined. This amounts to saying that t is a linearisation of the visible events in r. We shall drop the subscript E if no confusion will arise. If E is nac then f is unique.

We define the *traces* of E, denoted $tr(E)$, to be the set of all t such that $obs(t, r)$ for some run r of E. For $t \in tr(E)$ let $runs(t)$ denote the set of all runs r of E such that $obs(t, r)$.

Definition 5.1 Let E be an event structure and r a run of E.

1. r is *live* for an event $e \in E$ if $e \notin r$ and $r \cup \{e\}$ is a run. Denote the set of such events by $live(r)$.
2. r is *stable* if it is maximal with respect to hidden events, that is, $live(r)$ contains no hidden events.
3. r *refuses* a set $X \subseteq Act$ if r is stable and $X \cap \{\ell(e) \mid e \in live(r)\} = \emptyset$.

4. For $t \in Act^\star, X \in \mathcal{P}_f(Act)$, $(t, X) \in \mathit{fail}(E)$ iff there exists a run r such that $obs(t, r)$ and r refuses X.

Proposition 5.2 Let P be a finite CSP process. Then $\mathit{fail}(P) = \mathit{fail}(\mathcal{E}(P))$.

Now we turn our attention to our concurrency and conflict semantics. We define the counterpart of the $co()$ and $cf()$ relations, as well as additional relations $cp()$ of *compatibility*, $gco()$ of *guaranteed concurrency*, and $pre()$ of *causal precedence*. We then show that there exist connections between them. The complement of the relation $cf()$ is denoted by $\overline{cf}()$.

We shall need the following auxiliary notions. String prefix order is denoted by $\leq$. We say $s\ perm\ t$ iff the trace s is a permutation of t. $s \subseteq t$ denotes the multiset order; more formally, $s \subseteq t$ iff $\exists s'.s\ perm\ s' \leq t$. For $s, t \in Act^\star$, their *string difference* is defined inductively by:

1. $\varepsilon - s = \varepsilon$
2. $s - \varepsilon = s$
3. $a - a = \varepsilon$
4. $as - t \quad = \quad \begin{cases} s - (t - a) & \text{if } a \in t \\ a(s - t) & \text{otherwise} \end{cases}$

The *leftmost difference* $\mathrm{ld}(s, t)$ of traces $s, t \in Act^\star$ is defined to be $head(s - t)$ if $s - t \neq \varepsilon$, and undefined otherwise.

Definition 5.3 Let E be an event structure and $s, t \in tr(E)$.

1. $s\ cp(E)\ t$ iff there are $r \in runs(s), r' \in runs(t)$ such that $r\ cp\ r'$;
2. $s\ cf(E)\ t$ iff there are $r \in runs(s)$, $r' \in runs(t)$ such that $r \# r'$;
3. $s\ co(E)\ t$ iff there is $r \in runs(s) \cap runs(t)$;
4. $s\ gco(E)\ t$ iff $runs(s) = runs(t)$;
5. $s\ pre(E)\ t$ iff $s \subseteq t$ and $s\ \overline{cf}(E)\ t$.

Lemma 5.4 Let E be nac. If $s, t \in tr(E)$, r, r' are runs of E such that $s \subseteq t$, $obs(s, r)$, $obs(t, r')$ and $r\ cp\ r'$, then $obs(t, r \cup r')$.

We can now show that the $co()$ and $gco()$ relations are the natural counterparts of the Mazurkiewicz trace equivalence since they relate two different observations of the same run.

Proposition 5.5 Let E be nac. For any $s, t \in tr(E)$:

1. $s\ co(E)\ t$ iff $s\ cp(E)\ t$ and $s\ perm\ t$;
2. $s\ cp(E)\ t$ iff $s(t - s)\ co(E)\ t(s - t)$;
3. $s\ gco(E)\ t$ iff $s\ \overline{cf}(E)\ t$ and $s\ perm\ t$.

Proof: Follows from Lemma 5.4. $\square$

These last two results mean that the $cp(E)$, resp. the $gco(E)$, semantics may be derived from our $co(E)$, resp. $cf(E)$, semantics. Furthermore, the next result shows that the relation $pre()$ is a counterpart of the causal ordering on E, as well as the Mazurkiewicz trace preorder. Part (1) clearly corresponds to forward propagation of conflict.

Proposition 5.6 Let E be nac.

1. If $s\ cf(E)\ t$ and $t\ pre(E)\ u$ then $s\ cf(E)\ u$.
2. $pre(E)$ is a preorder.

Proof: (1) is by Lemma 5.4. (2) comes from (1). $\square$

Now, if $s\ gco(E)\ t$ then s and t should be in a sense indistinguishable:

Proposition 5.7 Let E be nac.

1. Suppose $u\ cf(E)\ sv$ and $s\ gco(E)\ t$. Then $u\ cf(E)\ tv$.
2. Suppose $(su, X) \in fail(E)$ and $s\ gco(E)\ t$. Then $(tu, X) \in fail(E)$.

The following result shows that there is a connection between conflict and failure semantics.

Proposition 5.8 Let E be nac.

1. Suppose $(s, X), (s, Y) \in fail(E)$ and $s\ \overline{cf}(E)\ s$. Then $(s, X \cup Y) \in fail(E)$.
2. Suppose $(s, \{a\}) \in fail(E)$, $t \in tr(E)$ and let $a = \mathrm{ld}(s, t)$. Then $s\ cf(E)\ t$.

Part (1) states that if a process is deterministic in the sense of not having self-conflict, then it is deterministic in the failures semantics (*i.e.* does not have essentially different refusals). A simple case of part (2) would be where $(b, \{a\}) \in fail(E)$ and $ba \in tr(E)$. We expect that there must be two conflicting runs, where a is either performed or refused. The result guarantees that $b\ cf(E)\ ba$ in accordance with this expectation.

Now we wish to establish that our two semantics ($co()$ and $cf()$) are sound under the event structure interpretation. This is shown by structural induction on finite processes using the definitions of the CSP operators on event structures as in [15]. We have not extended our work to event structures for recursive processes. The key issue is parallel composition, and (in the case of the $cf()$ semantics) our result depends on non-autoconcurrency.

We do not repeat the definition of the map from CSP to event structures given in [15]. The most difficult part of this is the definition of the parallel composition of two event structures. We adopt a slightly different version, which we denote $E_{1A} \parallel E_{2B}$ (or $E_1 \parallel E_2$ where A and B are understood). The full paper will contain the definition, which we omit from this version. It is similar to those given in [21] and [15] (though our operator is slightly different). Essentially the runs of $E_1 \parallel E_2$ are sets of pairs of events from E_1, E_2. If r is a run of $E_1 \parallel E_2$ then we can project r in a standard way to get runs $\pi_1(r)$, $\pi_2(r)$ of E_1, E_2 respectively.

Non-autoconcurrency is a particularly useful property, and so we would like the image of a CSP process to be a nac event structure. Various authors have argued against autoconcurrency, see *e.g.* [3]; for example, it is disallowed in trace theory. It turns out that the mixed parallel is the right one to consider since it gives rise to non-autoconcurrent event structures.

Proposition 5.9 Let P be a finite CSP process. Then $\mathcal{E}(P)$ is nac.

The only difficulty with the above result is in showing that if E_1, E_2 are nac then so is $E_1 \| E_2$. It is important to note that nac would fail if we allowed many-one renaming. In the full paper we shall consider the other parallel operators known from the literature.

Lemma 5.10 Let E_1, E_2 be nac. Let r, r' be runs of $E = E_1 \parallel E_2$. Then

$$r \# r' \quad \text{iff} \quad \pi_1(r) \#_1 \pi_1(r') \;\text{ or }\; \pi_2(r) \#_2 \pi_2(r')$$

Proof: (Remarks.) The "if" is straight from the definition of parallel composition. For "only if" we need nac. $\square$

The nac assumption is essential in the above result. For a counterexample, let $E_1 = \{e\}, E_2 = \{e', e''\}$, with $e' \; co_2 \; e''$. Let all events be labelled with a. So E_2 exhibits autoconcurrency. Now $\{(e, e')\}$ and $\{(e, e'')\}$ are both runs of $E_1 \parallel E_2$ and their projections do not conflict. However, $\{(e, e'), (e, e'')\}$ is not a run of $E_1 \parallel E_2$.

Proposition 5.11 If P is a finite CSP process, then $cf(P){=}cf(\mathcal{E}(P))$.

Proof: By structural induction on P. The case for parallel composition is handled by Lemma 5.10. We omit the checks for the other operators. $\square$

Lemma 5.12 Let E_1, E_2 be event structures, let $E = E_{1A} \parallel E_{2B}$, and let $s, s' \in tr(E)$. If $s\lceil A \; co(E_1) \; s'\lceil A$ and $s\lceil B \; co(E_2) \; s'\lceil B$ then $s \; co(E) \; s'$.

Proof: (Idea.) We take common runs for $s\lceil A$, $s'\lceil A$ and $s\lceil B$, $s'\lceil B$ and "knit" them together to form runs for E. $\square$

Proposition 5.13 If P is a finite CSP process, then $co(P){=}co(\mathcal{E}(P))$.

The import of Propositions 5.11 and 5.13, together with Proposition 5.2, is that our semantics, while simpler and more abstract, is "as good as" the event structure semantics. Thus, concurrency distinctions can be made at a high level of abstraction, without resorting to reasoning about event occurrences.

6 Further work

A further refinement of the semantics would be by adding causality information. The definition is similar to that of possible concurrency relation $co()$.

Another direction is to adopt a more discriminating notion of an observation, such as failures, instead of traces. Observations as failures are well motivated, since they can be explained in terms of experiments on machines. A generalization of the concurrency relation to work on failures instead of traces is straightforward, and it would lead to finer distinctions.

Finally, we would like to derive a complete axiomatization of the failure–concurrency, resp. failure–conflict, pairs analogous to the axioms for failure–divergence pairs [5]. It is relatively well known that any finite CSP process is equivalent to one obtained using 0, prefixing, $\sqcap$ and $\square$ [7]. The interesting point is that the parallel operator $\|$ is not used. We would anticipate that in our model, if there is a satisfactory axiomatization, the parallel operator would play an essential role.

7 Conclusion

We have proposed a framework of refinements of the standard failures model for CSP, the purpose of which was to weaken the expansion law to an inequality. We have shown that the additional information about concurrency, conflict and causality can be derived systematically from the syntax of processes, without the need for additional labelling of actions, as long as the relations are interpreted as representing the *possibility* (or, dually, the *guarantee*) of the given phenomenon. We have carried out the construction for a large subset of CSP, which includes hiding, with respect to the relations of possible concurrency and conflict. We have found that the laws of the "possible concurrency" model are essentially the same as the original CSP laws, the exception being the expansion law. The "possible conflict" (or "guaranteed concurrency") calculus is less well behaved, as it also fails the idempotency of choice. The resulting inequational calculi are only slightly more complex than the original failures calculus.

We believe that this framework is, to a large extent, independent of the choice of the language and model. However, we should point out that the CSP parallel is the right one to make the construction work.

Acknowledgements

We would like to thank Bill Roscoe, Rob van Glabbeek, Frits Vaandrager, Samson Abramsky and Mike Shields for helpful discussions on the subject. Bill Roscoe's contribution in particular was invaluable, in that he directed our attention to the significance and potential technical advantages of possible concurrency, an aspect we had rather neglected, being hampered at the time by a belief that guaranteed concurrency would be of more interest in applications.

References

[1] Luca Aceto and Uffe Engberg. Failure semantics for a simple process language with refinement. In *Foundations of Software Technology and Theoretical Computer Science*, pages 89–108. Springer-Verlag, 1991.

[2] J.C.M. Baeten and J.A. Bergstra. Non interleaving process algebra. In *CONCUR 93*, volume 715 of *Lecture Notes in Computer Science*, pages 308–323. Springer-Verlag, 1993.

[3] Bard Bloom and Marta Z. Kwiatkowska. Trade-offs in true concurrency: Pomsets and Mazurkiewicz traces. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 598 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[4] Gerard Boudol and Ilaria Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer-Verlag, 1989.

[5] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[6] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer-Verlag, 1985.

[7] Stephen Brookes. *A Model for Communicating Sequential Processes.* PhD thesis, Oxford University, 1983.

[8] P. Degano, R. de Nicola, and U. Montanari. Universal axioms for bisimulation. *Theoretical Computer Science*, 114:63–91, 1993.

[9] P. Degano and U. Montanari. Concurrent histories: A basis for observing distributed systems. *Journal of Computer and System Sciences*, 34:422–461, 1987.

[10] P. Godefroid and P. Wolper. A partial approach to model checking.

[11] M. Hennessy. Concurrent testing of processes. Technical Report 11/91, University of Sussex, 1991.

[12] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[13] P.W. Hoogers, H. Kleijn, and P.S. Thiagarajan. A trace semantics for petri nets. In *ICALP'92*, volume 623 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[14] Marta Z. Kwiatkowska and Iain C.C. Phillips. Concurrency and conflict in CSP. In G.L. Burn, S.J. Gay, and M.D. Ryan, editors, *Theory and Formal Methods 1993: Proceedings of the First Imperial College, Department of Computing, Workshop on Theory and Formal Methods*, Workshops in Computer Science, pages 209–225. Springer-Verlag, 1993.

[15] R. Loogen and U. Goltz. Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informaticae*, XIV:39–74, 1991.

[16] Antoni Mazurkiewicz. Basic notions of trace theory. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1989.

[17] M. Mukund and M. Nielsen. CCS, locations and asynchronous transition systems. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[18] M. Nielsen, G. Winskel, and G. Plotkin. Petri nets, event structures and domains. *Theoretical Computer Science*, 13:85–108, 1981.

[19] M.W. Shields. Deterministic asynchronous automata. In *Formal Methods in Programming.* North Holland, 1985.

[20] D. Taubner and W. Vogler. The step failures semantics. In *STACS'87*, Lecture Notes in Computer Science, pages 348–359. Springer-Verlag, 1987.

[21] F. Vaandrager. A simple definition for parallel composition of prime event structures. Technical Report CS-R8903, CWI, 1989.

[22] Glynn Winskel. Event structure semantics for CCS and related languages. In *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 561–567. Springer-Verlag, 1982.