CMU-CS-83-111

On the relationship of CCS and CSP

Stephen D. Brookes
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa.

March 1983

A version of this paper will appear in *Proceedings of ICALP 1983.*

The research reported in this paper was supported in part by funds from the Computer Science Department of Carnegie-Mellon University, and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. The views and conclusions contained in it are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

ON THE RELATIONSHIP OF CCS AND CSP

Stephen D. Brookes
Department of Computer Science
Carnegie-Mellon University
Pittsburgh
Pennsylvania 15213

Abstract.

This paper compares two models of concurrency, Milner's Calculus of Communicating Systems (CCS) and the *failures* model of Communicating Sequential Processes (CSP) developed by Hoare, Brookes and Roscoe. By adapting Milner's synchronisation trees to serve as notation for both CCS and CSP, we are able to define a representation mapping for CSP processes. We define an equivalence relation on synchronisation trees which corresponds precisely to the notion of ***failure equivalence.*** This equivalence relation identifies two trees if and only if the processes represented by the trees have identical failure sets. Milner's calculus is founded on a different notion, ***observation equivalence.*** We show how these two equivalences are related. Just as Milner's equivalence can be characterised as the smallest relation satisfying a set of axioms, we find a suitable set of axioms for the failures equivalence relation. This again makes explicit the differences between the two systems, as well as revealing that the semantic models underlying CCS and CSP are comparable.

1.0. Introduction.

This paper considers the similarities and differences between two abstract models of concurrent behaviour, Milner's synchronisation trees for CCS [1], and the failures model of CSP (Hoare, Brookes, Roscoe [2]). In order to make the paper relatively self-contained, we begin by listing the principal characteristics of the two systems. Milner's original formulation of his calculus introduced synchronisation trees, with arcs labelled by action names drawn from an alphabet Σ or by a special symbol τ standing for an invisible action; paths through a tree then correspond to a sequence of visible actions, possibly with some invisible actions occurring on the way. Each node of a tree defines a possible sequence of visible actions up to some moment, and the subtree rooted there represents a possible future behaviour. Milner defines a notion of behaviour for synchronisation trees and constructs an equivalence relation on trees known as ***observation equivalence.*** Terms in the language CCS can then be taken to denote equivalence classes of trees under observation equivalence.

In the failures model of Hoare, Brookes and Roscoe the behaviour of a process is defined in terms of the sequences of visible actions the process may perform, and the sets of actions the process may (as the result of making a nondeterministic decision) refuse to perform. A failure is simply a pair consisting of a finite sequence of visible actions possible for the process and a set of actions which the process may be able to refuse on the next step after this sequence. The behaviour of a process is then determined by its failure set. There is a natural partial order on behaviours which captures precisely the notion of nondeterminism and turns the set of all process behaviours into a complete semi-lattice. Terms in the language CSP can then be taken to denote failure sets.

We will give an alternative formulation of processes equivalent to the failures definition. The new version is designed in order to facilitate comparison with CCS. Specifically, we define a mapping from CSP to synchronisation trees, and an equivalence relation (called failure equivalence) on trees which reflects the failure semantics of processes. Two processes have the same failure sets if and only if the trees representing them are identified by the failure equivalence relation. We also define operations on synchronisation trees which mirror the process operations of CSP. This leads to a discussion of which CSP operations are definable in terms of Milner's CCS operations. We also show that the failure equivalence relation is the relation characterised by a set of axioms, and compare these axioms with the defining axioms of observation equivalence.

1.1. Milner's synchronisation trees.

This section contains a summary of the definitions and results of Milner. More details can be found in [1]. We begin with a set Σ of *actions*, also known as *events*. This set is called the *alphabet*. There is also a special symbol τ, which does not belong to Σ : τ represents an *invisible* action. The set $\Sigma \cup \{\tau\}$ will be called the *extended alphabet*, and we use meta-variables a, b to range over the alphabet, and λ, μ to range over the extended alphabet. The meta-variables s, t, u range over finite sequences of events, and w ranges over finite sequences of extended events.

A *synchronisation tree* S is an rooted, unordered, finitely branching tree all of whose arcs are labelled with either τ or an event. We use the notation

$$\sum_{i=1}^{n} \mu_i T_i$$

for the tree whose initial arcs are labelled $\mu_1, \ldots, \mu_n$, and which has subtrees $T_1, \ldots, T_n$ at the ends of these arcs. The trivial tree with no arcs is denoted *NIL*, and the result of joining two trees S and T at their roots is denoted $S + T$. The meta-variables S, T, U range over trees. The ***branches*** of a tree are defined in the usual way. Note that NIL has no non-trivial branches, and the non-trivial branches of $S + T$ are either branches of S or of T. The following axioms reflect our assumption that a tree is uniquely determined by its set of branches.

PROPOSITION 1.1.1. *Addition is commutative, idempotent and associative; NIL is an additive identity element.*

$$\begin{array}{ll} (A1) & S+T=T+S \\ (A2) & (S+T)+U=S+(T+U) \\ (A3) & S+S=S \\ (A4) & S+NIL=S \end{array}$$

If S has a branch of the form wT, we write

$$S \xrightarrow{w} T,$$

and say that S has a w-branch (to T.) As far as an observer of a tree is concerned, the τ actions are invisible; we use the notation $w \,/\, \tau$ for the sequence of visible actions obtained by deleting all occurrences of τ from w, and write

$$S \stackrel{t}{\Longrightarrow} T$$

when S has a branch to T on which the sequence of visible actions is t; we say that S has a *t-derivation* (to T). A t-derivation represents a possible behaviour in which the sequence of visible actions t occurs and where the behaviour thereafter may be any consistent with T. The behaviour of a process will be modelled by a synchronisation tree, and two processes will be distinguishable only if their possible derivations differ. In making this more precise, Milner defines a sequence of equivalence relations $\{ \approx_n \mid n \geq 0 \}$ on trees, with the idea being that the n^{th} relation represents equivalence up to depth n.

DEFINITION 1.1.2. The equivalence relations $\approx_n$ $(n \geq 0)$ are defined by:

(i) $S \approx_0 T$ for all S, T.
(ii) $S \approx_{n+1} T$ iff, for all $s \in \Sigma^*$,
  (a) $S \stackrel{s}{\Longrightarrow} S' \Rightarrow \exists T'.T \stackrel{s}{\Longrightarrow} T' \,\&\, S' \approx_n T'$
  (b) $T \stackrel{s}{\Longrightarrow} T' \Rightarrow \exists S'.S \stackrel{s}{\Longrightarrow} S' \,\&\, S' \approx_n T'$.

It is clear that each of these relations is indeed an equivalence, and that they form a decreasing chain of finer and finer relations:

$$\approx_{n+1} \subset \approx_n, \qquad \text{for all } n.$$

Milner regards two trees as observationally equivalent if and only if they cannot be distinguished by any finite experiment; this is the case when no $\approx_n$ relation can distinguish between them. This motivates the following definition.

DEFINITION 1.1.3. Two trees S and T are observationally equivalent iff, for all $n \geq 0$, $S \approx_n T$. The observation equivalence relation $\approx$ is defined by:

$$\approx = \bigcap_{n=0}^{\infty} \approx_n .$$

It is obvious that observation equivalence is indeed an equivalence relation:

(i) $S \approx S$
(ii) $S \approx T \Rightarrow T \approx S$
(iii) $S \approx T \approx U \Rightarrow S \approx U$.

Milner notes the following laws of observation equivalence [1]. They are easily verified; one uses induction on n to prove that the appropriate pairs of trees are n-equivalent for all n.

PROPOSITION 1.1.4. *The following laws hold for observation equivalence:*

(1) $S + \tau S \approx \tau S$
(2) $\tau S \approx S$
(3) $\mu S + \mu(\tau S + T) \approx \mu(\tau S + T)$

Milner also defines an inference rule, known as *guarded inference.*

PROPOSITION 1.1.5. *The following inference rule, (R), is valid:*

$$(R) \qquad \frac{S \approx T}{\mu S + U \approx \mu T + U}.$$

As Milner shows, there is a sense in which these laws and inference rule *characterize* observation equivalence, at least on *finite* trees. One can use these laws to prove every true equivalence on finite trees, provided one allows use of laws (1) and (3), (A1)–(A4), in any additive context. Law (2), however, is not valid in all contexts, so its use must be restricted. The following proposition states this fact more precisely.

PROPOSITION 1.1.6. *The following set of axioms, together with rule (R) and laws (A1)–(A4), is complete for observation equivalence of finite trees.*

$$\begin{array}{ll} (M1) & S + \tau S + T \approx \tau S + T \\ (M2) & \tau S \approx S \\ (M3) & \mu S + \mu(\tau S + T) + U \approx \mu(\tau S + T) + U \end{array}$$

Proof. These are (essentially) Milner's τ-laws, and the completeness result is stated in [1]. ∎

These results for CCS are well known. Later we will show that similar results can be obtained for CSP, with a different axiomatic system.

Milner uses synchronisation trees and the observation equivalence relation in constructing a mathematical model of concurrent processes. He introduces a simple language, called CCS, whose terms can be taken to denote (equivalence classes of) synchronisation trees. For our purposes, the terms in this language can be thought of as being generated by the following grammar:

$$S ::= NIL \mid aS \mid S_1 + S_2 \mid S_1 \mid S_2 \mid S\backslash a \mid S[a \setminus b].$$

We have already dealt with the first three forms. Milner calls $S \mid T$ the *composition* of S and T, and for trees

$$S = \sum_{i=1}^{n} \lambda_i S_i,$$
$$T = \sum_{i=1}^{m} \mu_i T_i,$$

with $\lambda_i, \mu_j \in \Sigma \cup \{\tau\}$, the composition is defined by

$$S \mid T = \sum_{i=1}^{n} \lambda_i (S_i \mid T) + \sum_{j=1}^{m} \mu_j (S \mid T_j) + \sum_{\lambda_i = \overline{\mu_j}} \tau(S_i \mid T_j).$$

Here the events a and $\bar{a}$ are *matching* or *complementary* actions. It will simplify our presentation without losing any generality to assume that the only actions which have complements are visible actions, and that $a = \bar{a}$ for all visible actions a. The final two types of CCS process are interpreted thus, using the same notation as above for S :

$$S\backslash b = \sum_{\lambda_i \neq b} \lambda_i(S_i\backslash b),$$

$$S[a \setminus b] = \sum_{i=1}^{n} \lambda_i[a \setminus b]S_i[a \setminus b],$$

where for an event μ, $\mu[a \setminus b]$ is a if $\mu = b$ and μ otherwise. These operations are called ***restriction*** ($\backslash b$) and ***relabelling*** ($[a \setminus b]$) by Milner. Restricting prunes away branches involving the particular event b, while relabelling replaces all occurrences of one label by another. Note that these operations are defined recursively, and these definitions can be thought of as ***expansion theorems*** which allow a term involving composition, relabelling or restriction to be manipulated into a summation form. Milner shows that addition of these expansion laws to the logical system of Proposition 1.1.6 produces a complete system for the full language of (finite) CCS terms.

It is important to note that Milner's observation equivalence is not quite a ***congruence*** with respect to his operations. In particular, it is not an additive congruence, in that there are trees which are observationally equivalent but can be put in a context where they are no longer equivalent. For example, it is always true that $S \approx \tau S$, but obviously not generally true that $S + T \approx \tau S + T$. Addition is the only operation which causes problems here; all other CCS operations preserve observation equivalence.

1.2. The failures model of CSP.

In the failures model of process behaviour, a process is characterised as a ***failure set.*** Each possible failure of a process represents a finite piece of behaviour in which the process has engaged in a sequence of visible actions up to some moment and has since then refused to participate in some set of actions, *i.e.* the process has ***refused*** a set of actions. This refusal comes about as the result of an autonomous decision by the process, and models the possibility of nondeterministic behaviour. Failures are intended to capture precisely the situations in which a process can ***deadlock.***

Again we begin with a set Σ of ***events,*** and events stand for process actions which are visible to the process's environment. In the CSP model we are thinking of events as standing for synchronised ***communications*** or ***interactions*** between a process and its environment. Instead of using τ as a special symbol for an unobservable action, and allowing occurrences of τ to represent nondeterministic behaviour, the presence of nondeterminism manifests itself as follows. After each finite sequence of visible actions, a process has a set of ***refusal sets*** which represents the possible consequences, for the next step, of the various nondeterministic decisions available to the process. We imagine that a nondeterministic decision has the effect of ***removing*** a set of events from the set of actions in which the process might have participated on the next step. In other words, each nondeterministic decision restricts the process's future behaviour. Thus it is appropriate to represent this effect as a ***refusal set.***

A failure is a ***pair*** consisting of a sequence s of events and a set X of events. We will refer to s as the ***trace*** and X as the ***refusal set.*** Intuitively, if a particular failure (s, X) is possible for a process then the process may, once it has performed the sequence s, refuse to participate in any event in X on the next step. Thus we say that the process may do s and then refuse X. If (s, X) is a possible failure of a process and the process is run in an environment in which the sequence of events s is allowed and then the environment only allows events in X as the next step, there is a possibility of deadlock: the process can refuse all of the events which the environment is willing to perform next.

A process P will be characterised as a set of failures, or (equivalently) as a relation between traces and refusal sets. The domain dom(P) of this relation will define the trace set of the process.

DEFINITION 1.2.1. A process is a set of failures P satisfying:

$$\begin{array}{ll}
\text{(P1)} & (\langle\rangle, \emptyset) \in P \\
\text{(P2)} & (st, \emptyset) \in P \Rightarrow (s, \emptyset) \in P \\
\text{(P3)} & X \subseteq Y \;\&\; (s, Y) \in P \Rightarrow (s, X) \in P \\
\text{(P4)} & (s, X) \in P \;\&\; s\langle b\rangle \notin \mathrm{dom}(P) \Rightarrow (s, X \cup \{b\}) \in P
\end{array}$$

The above definition says that the traces of a process form a non-empty set (P1), which is also prefix-closed (P2). If P can refuse a set Y at some stage then it can also refuse any subset X of Y (P3). An *impossible* event can always be included in a refusal set (P4). These conditions are intuitively appealing, given our model of behaviour.

DEFINITION 1.2.2. For any set P of failures,

$$\begin{array}{lr}
\text{(i)} & \mathrm{traces}(P) = \{ s \mid (s, \emptyset) \in P \} \\
\text{(ii)} & \mathrm{refusals}(P) = \{ X \mid (\langle\rangle, X) \in P \} \\
\text{(iii)} & \mathrm{initials}(P) = \{ a \mid \langle a\rangle \in \mathrm{traces}(P) \}
\end{array}$$

If P and Q are two processes such that $P \supseteq Q$, then every possible failure of Q is also possible for P. Intuitively this means that it is possible for P to behave like Q, but it may also be the case that P can behave in a manner impossible for Q, either by refusing or performing more than Q could at the same stage. In such circumstances we say that P is more nondeterministic than Q, and write $P \sqsubseteq Q$. It is easy to see that processes are partially ordered by this relation. In fact the set of processes becomes a *complete semi-lattice* under this ordering.

PROPOSITION 1.2.3. *Processes, ordered by $\sqsubseteq$, form a complete semi-lattice; that is, $\sqsubseteq$ is a partial order, there is a least element (known as CHAOS), every non-empty set of processes has a greatest lower bound, and every directed set of processes has a least upper bound.*

Proof. The union of any non-empty set of processes is again a process, and the intersection of any directed set of processes is a process. The bottom element is CHAOS $= \Sigma^* \times \mathcal{P}(\Sigma)$. Details can be found in [2]. ∎

In [2] we introduced a denotational semantics for a simplified version of Hoare's CSP language, in which CSP processes were identified with failure sets. We defined a set of operations on failure sets which correspond to the syntactic constructs of the language, and showed that all of these operations are continuous with respect to the nondeterminism ordering. This fact justified our use in [2] of recursively defined processes, since least fixed points of continuous functions on complete semi-lattices exist. For the purposes of this paper, the following is a summary of the relevant work.

The syntax of the language is simple. The syntactic category of *processes* P is defined thus:

$$\begin{aligned} P ::= \mathrm{STOP} \mid (a \rightarrow P) \mid P \sqcap P \mid P \,\Box\, P \mid P \| P \\ \mid P ||| P \mid P \,/\, b \end{aligned}$$

STOP is intended to be a process which is unable to perform any action; this corresponds to *deadlock*. We refer to the other syntactic constructs as *prefixing*, *unkind choice*, *kind choice*, *strict parallel composition*, *interleaving* and *hiding*. The result of prefixing an event a to a process P is a process which must initially perform a and then behaves like P. The difference between the two forms of *choice* operation manifests itself only on the initial step: $P \,\Box\, Q$ is not allowed to refuse an event unless both constituent processes refuse it; $P \sqcap Q$ can refuse an event if either of the constituents chooses to do so. In both forms of choice, once an event has occurred, only one of the constituent processes is still active. The reader familiar with Hoare's original language might recognise that a kind choice corresponds to a guarded command in which all the guards are communications, so that the process's environment must be consulted before determining which guard to pass; likewise, an unkind choice represents the case where all guards are purely boolean, so that a guard can be passed without consulting the environment. The process $P \| Q$ is a form of parallel composition in which each event occurs only if both constituents perform it together; this obviously represents a very tightly coupled form of parallelism. In contrast, the interleaving of two processes allows them to execute events independently of each other, so that the traces of $P|||Q$ will be obtained by interleaving traces from P and Q. The hiding operator renders an event invisible to the environment, and allows its action to take place nondeterministically. For further details the reader is referred to [2].

The semantic function F maps a process P to its failure set $F(P)$, and is defined by a structural induction in the usual way. Thus for the "terminal" cases (*i.e.* STOP) we define the failures explicitly while in general the failures of P are built up from the failures of its immediate syntactic components.

DEFINITION 1.2.4. The failures semantic function F is defined by structural induction as follows:

$$\begin{aligned}
F[\![\mathrm{STOP}]\!] &= \{ (\langle\rangle, X) \mid X \subseteq \Sigma \} \\
F[\![a \rightarrow P]\!] &= \{ (\langle\rangle, X) \mid a \not\in X \} \cup \{ (\langle a\rangle s, X) \mid (s, X) \in F[\![P]\!] \} \\
F[\![P_1 \sqcap P_2]\!] &= F[\![P_1]\!] \cup F[\![P_2]\!] \\
F[\![P_1 \,\Box\, P_2]\!] &= \{ (\langle\rangle, X) \mid (\langle\rangle, X) \in F[\![P_1]\!] \cap F[\![P_2]\!] \} \cup \{ (s, X) \mid s \neq \langle\rangle \;\&\; (s, X) \in F[\![P_1]\!] \cup F[\![P_2]\!] \} \\
F[\![P_1 \| P_2]\!] &= \{ (s, X \cup Y) \mid (s, X) \in F[\![P_1]\!] \;\&\; (s, Y) \in F[\![P_2]\!] \} \\
F[\![P_1 ||| P_2]\!] &= \{ (u, X) \mid \exists s, t. (s, X) \in F[\![P_1]\!] \;\&\; (t, X) \in F[\![P_2]\!] \;\&\; u \in \mathrm{merge}(s, t) \} \\
F[\![P \,/\, b]\!] &= \{ (s \,/\, b, X) \mid (s, X \cup \{ b \}) \in F[\![P]\!] \}
\end{aligned}$$

Notice that the intuition behind each syntactic operation is captured in this semantic definition. Thus, for instance, X is a refusal of $(a \to P)$ iff X does not contain a : this process cannot refuse a. And the difference between the two forms of choice operation is exemplified by the fact that the process

$$(a \to \mathrm{STOP}) \Box (b \to \mathrm{STOP})$$

must initially perform either a or b, and cannot commit itself to one of these events over the other; although $\{a\}$ is a refusal of $(b \to \mathrm{STOP})$, the other component process $(a \to \mathrm{STOP})$ cannot refuse a. On the other hand, the process

$$(a \to \mathrm{STOP}) \sqcap (b \to \mathrm{STOP})$$

can choose arbitrarily whether or not to allow a or b, and $\{a\}$ and $\{b\}$ are refusals of this process; the fact that one or other of the events must occur initially is reflected in the process's inability to refuse $\{a, b\}$.

We have given here the definition of hiding appropriate for finite processes: when the language is extended to allow infinite processes there are several alternative definitions of hiding, all of which agree on finite processes; it would not be appropriate to give details here, since we are only dealing with finite terms. More details can be found in [2].

In the next section we will link the failure set semantics with the synchronisation tree model, by defining an equivalence relation on synchronisation trees which naturally represents failure sets. This will enable us to define a semantic mapping from CSP to (equivalence classes of) trees which matches precisely the failure set semantics. In doing so, we will define tree operations analogous to the syntactic CSP operations.

1.3. The failure equivalence relation on synchronisation trees.

We begin by defining some basic attributes of synchronisation trees. The ***traces*** of a tree are simply the sequences of visible actions that appear on the branches of the tree, and (similarly) the initials are the visible events appearing first on the branches. A tree can ***refuse*** a set X of events if there is a τ-branch from the root to a subtree whose initials are disjoint from X. Equivalently, S can refuse X if it has an invisible transition to a tree which has no x-actions, for all $x \in X$. Finally, the pair (s, X) is a ***failure*** of T if T has an s-derivation to a subtree whose initials are disjoint from X.

DEFINITION 1.3.1. For any synchronisation tree S,

(i) $\mathrm{traces}(S) = \{ s \mid \exists T.\, S \overset{s}{\Longrightarrow} T \}$
(ii) $\mathrm{initials}(S) = \{ a \mid \langle a \rangle \in \mathrm{traces}(S) \}$
(iii) $\mathrm{refusals}(S) = \{ X \mid \exists T.\, S \overset{\langle\rangle}{\Longrightarrow} T \,\&\, X \cap \mathrm{initials}(T) = \emptyset \}$
(iv) $\mathrm{failures}(S) = \{ (s, X) \mid \exists T.\, S \overset{s}{\Longrightarrow} T \,\&\, X \cap \mathrm{initials}(T) = \emptyset \}$

The ***failure equivalence relation*** on trees identifies two trees if and only if they have identical failure sets. The formal definition is:

DEFINITION 1.3.2. Failure equivalence is the relation $\equiv$ on trees given by

$$S \equiv T \quad \Leftrightarrow \quad \text{failures}(S) = \text{failures}(T).$$

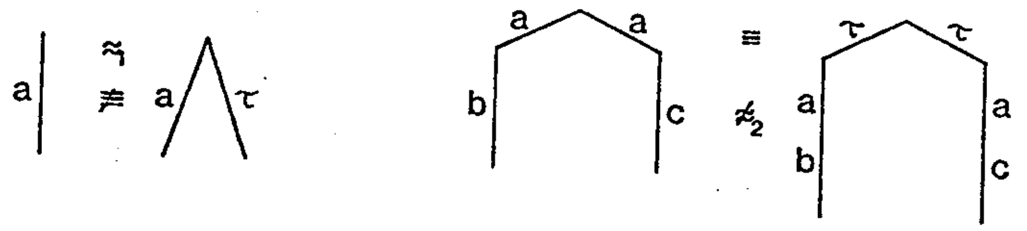
It is clear that this is an equivalence relation. From the definition, a comparison with the definitions of Milner's first and second relations shows that:

$$\begin{aligned}
&(1) \quad S \approx_1 T \quad \Leftrightarrow \quad \text{for all } s \in \Sigma^*, \\
&\qquad\qquad \exists S'.\, S \overset{s}{\Longrightarrow} S' \Leftrightarrow \exists T'.\, T \overset{s}{\Longrightarrow} T'
\end{aligned}$$

$$\begin{aligned}
&(2) \quad S \equiv T \quad \Leftrightarrow \quad \text{for all } s \in \Sigma^* \text{ and } X \subseteq \Sigma, \\
&\qquad\qquad \exists S'.\, S \overset{s}{\Longrightarrow} S' \;\&\; X \cap \text{initials}(S') = \emptyset \\
&\qquad\qquad \Leftrightarrow \exists T'.\, T \overset{s}{\Longrightarrow} T' \;\&\; X \cap \text{initials}(T') = \emptyset
\end{aligned}$$

$$\begin{aligned}
&(3) \quad S \approx_2 T \quad \Leftrightarrow \quad \text{for all } s \in \Sigma^*, \text{ and } U \subseteq \Sigma^*, \\
&\qquad\qquad \exists S'.\, S \overset{s}{\Longrightarrow} S' \;\&\; \text{traces}(S') = U \\
&\qquad\qquad \Leftrightarrow \exists T'.\, T \overset{s}{\Longrightarrow} T' \;\&\; \text{traces}(T') = U
\end{aligned}$$

Thus it follows that failure equivalence implies trace equivalence ($\approx_1$) and is implied by $\approx_2$. Hence, failure equivalence is a weaker relation than observation equivalence, because it makes more identifications and cannot distinguish between some pairs of trees which observation equivalence separates. Moreover, the failure relation is distinct from $\approx_1$ and $\approx_2$, as shown by the example:



PROPOSITION 1.3.3. *Failure equivalence lies between Milner's first and second equivalences, and is implied by observation equivalence:*

$$\approx_1 \supset \; \equiv \; \supset \; \approx_2 \; \supset \; \approx .$$

It is clear that we can define a pre-order $\sqsubseteq$ on trees which corresponds precisely to the nondeterminism ordering on processes, as follows.

DEFINITION 1.3.4. The pre-order $\sqsubseteq$ on trees is defined by

$$S \sqsubseteq T \quad \Leftrightarrow \quad \text{failures}(T) \subseteq \text{failures}(S).$$

COROLLARY 1.3.5. *For any pair of trees S and T,*

$$S \equiv T \quad \Leftrightarrow \quad S \sqsubseteq T \;\&\; T \sqsubseteq S.$$

Failure equivalence, like observation equivalence, can be axiomatized. We begin with a set of true equivalence laws and a valid rule of inference.

PROPOSITION 1.3.6. *The following laws hold for failure equivalence:*

$$\begin{array}{ll}
(B1) & S + \tau T + U \equiv \tau(S+T) + \tau T + U \\
(B2) & \tau S \equiv S \\
(B3) & \mu S + \mu T + U \equiv \mu(\tau S + \tau T) + U \qquad (\mu \in \Sigma \cup \{\tau\}) \\
(B4) & \tau(\mu S + T) + \tau(\mu S' + T') \equiv \tau(\mu S + \mu S' + T) + \tau(\mu S + \mu S' + T')
\end{array}$$

Proof. Verify for each law that the failures of the left-hand and right-hand side are identical. The details are omitted, as the proof is straightforward. ∎

PROPOSITION 1.3.7. *The following inference rule (R) is valid:*

$$(R) \quad \frac{S \equiv T}{\mu S + U \equiv \mu T + U} \qquad (\mu \in \Sigma \cup \{\tau\})$$

Note that we use the same name for this rule as for the corresponding rule in Milner's system.

As with the corresponding result for Milner's equivalence, these axioms and inference rule are complete for establishing equivalence of finite trees, provided we again allow use of the laws of Proposition 1.1.1 in any context. The proof that this system is complete is based on a simple algorithm for converting a tree to a "normal form," in which the arcs at each node have a special property known as *uniformity:* either the labels on the arcs are distinct and visible (*i.e.* not τ), or all the arcs are labelled with τ. It is clear that one can use (B1) and (B3) at a non-uniform node to produce uniformity; in each case any non-uniformity is pushed deeper into the tree, so that this method will definitely terminate on finite depth trees. One then shows that this process can be continued until the tree is in a *pre-normal form,* in which successive τ arcs have been collapsed down to single arcs (using (B3) again), and where the tree is "convex." Convexity can best be explained by observing that in a uniform tree each node without τ arcs corresponds to a sequence of visible events (the visible path from the root) and, conversely, each visible sequence s of actions corresponds to a set of such s-nodes (because the same sequence of actions may occur on different paths). Say that the set X of actions appears at an s-node of T if one of the s-nodes has arcs labelled by the events in X. The convexity condition is that, for each s, whenever X and Z appear at (different) s-nodes and $X \subseteq Y \subseteq Z$, then Y also appears at some s-node; and that whenever X and Y appear at some s-node so does $X \cup Y$. Axioms (B1) and (B3) are used to fill out a tree in this way. Finally one proves that any tree in pre-normal form can be converted, using (B4), into a normal form in which, for each sequence s all s-nodes have identical trees attached and these trees are also in normal form. Then it is easy to show that failure equivalence on trees in normal form is essentially the identity relation. Thus, each tree can be proven equivalent to a *unique* tree in full normal form; this establishes completeness of the system. The full proof is included in the appendix.

PROPOSITION 1.3.8. *The axiom system consisting of laws (A1)-(A4), (B1)-(B4), and rule (R) is complete for failure equivalence on finite trees.*

Proof. See Appendix. ∎

The following property of $\sqsubseteq$ enables us to obtain a complete proof system for the failures preorder on finite trees, by modifying the above axiom system.

PROPOSITION 1.3.9. ***For all trees*** S, T,

$$S \sqsubseteq T \quad \Leftrightarrow \quad \tau S + \tau T \equiv S.$$

We regard each of the above axioms for $\equiv$ as a pair of axioms involving $\sqsubseteq$, in the obvious way. We replace rule (R) by the rule given below:

$$(R') \qquad \frac{S \sqsubseteq T}{\mu S + U \sqsubseteq \mu T + U}.$$

And we add enough axioms and rules to give us that $\sqsubseteq$ is a pre-order satisfying Proposition 1.3.9:

$$\begin{array}{ll} (O1) & S \sqsubseteq T \sqsubseteq S \Leftrightarrow S \equiv T \\ (O2) & S \sqsubseteq T \sqsubseteq U \Rightarrow S \sqsubseteq U \\ (O3) & \quad S \sqsubseteq T \Leftrightarrow \tau S + \tau T \equiv S \end{array}$$

The proof of completeness for the earlier system can easily be modified to establish the truth of the following proposition.

PROPOSITION 1.3.10. ***The proof system generated by axioms*** $(A1)$-$(A4)$, $(B1)$-$(B4)$, $(O1)$-$(O3)$ ***and rule*** (R'), ***is complete for the failure pre-order on finite trees.***

1.4. Mapping CSP to synchronisation trees.

Now we can define a mapping from CSP syntax to synchronisation trees. If P is a CSP process then $\mathcal{T}[\![P]\!]$ will be a synchronisation tree having the same failure set as P. This will mean that two CSP processes have the same meaning in the failure set semantics if and only if their images under $\mathcal{T}$ are equivalent. The mapping $\mathcal{T}$ is defined by structural induction on the syntax, as usual.

DEFINITION 1.4.1. The map $\mathcal{T}$ from CSP to synchronisation trees is given by the following clauses:

$$\begin{aligned} \mathcal{T}[\![\mathrm{STOP}]\!] &= NIL \\ \mathcal{T}[\![a \to P]\!] &= a\mathcal{T}[\![P]\!] \\ \mathcal{T}[\![P_1 \sqcap P_2]\!] &= \tau\mathcal{T}[\![P_1]\!] + \tau\mathcal{T}[\![P_2]\!] \\ \mathcal{T}[\![P_1 \,\Box\, P_2]\!] &= \mathcal{T}[\![P_1]\!] \,\Box\, \mathcal{T}[\![P_2]\!] \\ \mathcal{T}[\![P_1 \| P_2]\!] &= \mathcal{T}[\![P_1]\!] \| \mathcal{T}[\![P_2]\!] \\ \mathcal{T}[\![P_1 ||| P_2]\!] &= \mathcal{T}[\![P_1]\!] ||| \mathcal{T}[\![P_2]\!] \\ \mathcal{T}[\![P \,/\, b]\!] &= \mathcal{T}[\![P]\!][\tau \setminus b] \end{aligned}$$

where the tree operations $\Box$, $\|$, $|||$ are defined so that for the trees

$$S = \sum_{i=1}^{n} a_i S_i + \sum_{i=1}^{N} \tau S_i',$$

$$T = \sum_{j=1}^{m} b_j T_j + \sum_{j=1}^{M} \tau T_j',$$

we have

$$S \Box T = \sum_{i=1}^{n} a_i S_i + \sum_{j=1}^{m} b_j T_j + \sum_{i=1}^{N} \tau(S_i' \Box T) + \sum_{j=1}^{M} \tau(S \Box T_j')$$

$$S \| T = \sum_{a_i = b_j} a_i(S_i \| T_j) + \sum_{i=1}^{N} \tau(S_i' \| T) + \sum_{j=1}^{M} \tau(S \| T_j')$$

$$S ||| T = \sum_{i=1}^{n} a_i(S_i ||| T) + \sum_{j=1}^{m} b_j(S ||| T_j) + \sum_{i=1}^{N} \tau(S_i' ||| T) + \sum_{j=1}^{M} \tau(S ||| T_j')$$

and the tree $S[\tau \setminus b]$ denotes the result of replacing every label b by τ in S. This is an instance of Milner's relabelling operation.

As examples, we can see that the trees representing the processes

$$\begin{array}{lccc} & (a \rightarrow \mathrm{STOP}) \Box (b \rightarrow \mathrm{STOP}) & \text{and} & (a \rightarrow \mathrm{STOP}) \sqcap (b \rightarrow \mathrm{STOP}) \\ \text{are} & aNIL + bNIL & \text{and} & \tau aNIL + \tau bNIL. \end{array}$$

To show that these tree operations do correspond to the original CSP operations, first we establish some results on the way transitions of a composite tree are built up from the transitions of the component trees.

Firstly, it is obvious that the following hold:

$$aS \xRightarrow{a} S$$
$$\tau S + \tau T \xRightarrow{\langle\rangle} S$$
$$\tau S + \tau T \xRightarrow{\langle\rangle} T$$

The behaviour of the tree operation $\Box$ is expressed by:

$$S \xRightarrow{\langle\rangle} S',\ T \xRightarrow{\langle\rangle} T' \ \Rightarrow\ (S \Box T) \xRightarrow{\langle\rangle} (S' \Box T'),$$
$$S \xRightarrow{s} S',\ s \neq \langle\rangle \ \Rightarrow\ (S \Box T) \xRightarrow{s} S',$$
$$T \xRightarrow{t} T',\ t \neq \langle\rangle \ \Rightarrow\ (S \Box T) \xRightarrow{t} T'.$$

For parallel composition we have:

$$S \xRightarrow{s} S',\ T \xRightarrow{s} T' \ \Rightarrow\ (S \| T) \xRightarrow{s} (S' \| T').$$

For interleaving:

$$S \xRightarrow{s} S',\ T \xRightarrow{t} T', u \text{ merges } s \,\&\, t \ \Rightarrow\ (S ||| T) \xRightarrow{u} (S' ||| T').$$

Finally, let $s \,/\, b$ denote the result of deleting all occurrences of b from the sequence s. Then

$$S \xRightarrow{s} S' \ \Rightarrow\ S[\tau \setminus b] \xRightarrow{s/b} S'[\tau \setminus b].$$

These facts can be used to prove the following result.

PROPOSITION 1.4.3. *The mapping $\mathcal{T}$ respects failure sets, in that for all processes P, we have*

$$\mathit{failures}(\mathcal{T}[\![P]\!]) = F[\![P]\!].$$

Proof. By structural induction on P. The base case is trivial, since $\mathcal{T}[\![\mathrm{STOP}]\!] = \mathit{NIL}$ and NIL has no non-trivial derivations. We give details for only two of the inductive steps.

Case 1. When $P = Q \| R$, let S and T be the trees representing Q and R respectively. The inductive hypothesis is that

$$\begin{aligned}\mathrm{failures}(S) &= F[\![Q]\!],\\ \mathrm{failures}(T) &= F[\![R]\!].\end{aligned}$$

We must prove that

$$\mathrm{failures}(S\|T) = F[\![Q\|R]\!].$$

But (s, X) is a failure of $S\|T$ if and only if there is a U such that

$$(S\|T) \overset{s}{\Longrightarrow} U \;\&\; X \cap \mathrm{initials}(U) = \emptyset.$$

From the definition of parallel composition on trees there must be some trees S' and T' such that

$$U = (S'\|T') \;\&\; S \overset{s}{\Longrightarrow} S' \;\&\; T \overset{s}{\Longrightarrow} T'.$$

But

$$\mathrm{initials}(S'\|T') = \mathrm{initials}(S') \cap \mathrm{initials}(T').$$

Let

$$\begin{aligned}Y &= X - \mathrm{initials}(S')\\ Z &= X - \mathrm{initials}(T').\end{aligned}$$

Then $X = Y \cup Z$ and by definition,

$$(s, Y) \in \mathrm{failures}(S) \;\&\; (s, Z) \in \mathrm{failures}(T).$$

By inductive hypothesis, this gives

$$(s, Y) \in F[\![P]\!] \;\&\; (s, Z) \in F[\![Q]\!],$$

and hence $(s, Y \cup Z) = (s, X) \in F[\![P\|Q]\!]$.

Case 2. For the case $P = Q \,\square\, R$, we may again assume the inductive hypothesis that

$$\begin{aligned}\mathrm{failures}(S) &= F[\![Q]\!],\\ \mathrm{failures}(T) &= F[\![R]\!].\end{aligned}$$

where S and T represent P and Q. We need to show that $\mathrm{failures}(S \,\square\, T) = F[\![P \,\square\, Q]\!]$. Consider first the refusals of $S \,\square\, T$. By definition of $\square$ on trees,

$$S \,\square\, T \overset{\langle\rangle}{\Longrightarrow} U \;\Leftrightarrow\; U = S' \,\square\, T' \;\&\; S \overset{\langle\rangle}{\Longrightarrow} S' \;\&\; T \overset{\langle\rangle}{\Longrightarrow} T'$$

for some trees S' and T'. Since initials($S \,\Box\, T$) is easily seen to be the union of initials(S) and initials(T), this gives

$$(\langle\rangle, X) \in \text{failures}(S \,\Box\, T) \;\Leftrightarrow\; (\langle\rangle, X) \in \text{failures}(S) \cap \text{failures}(T).$$

Hence, by the inductive hypothesis,

$$\begin{aligned}(\langle\rangle, X) \in \text{failures}(S \,\Box\, T) &\Leftrightarrow (\langle\rangle, X) \in \mathcal{F}[\![P]\!] \cap \mathcal{F}[\![Q]\!]\\ &\Leftrightarrow (\langle\rangle, X) \in \mathcal{F}[\![P \,\Box\, Q]\!].\end{aligned}$$

Finally, when $s \neq \langle\rangle$, we have

$$S \,\Box\, T \overset{s}{\Longrightarrow} U \;\Leftrightarrow\; \text{either } S \overset{s}{\Longrightarrow} U \text{ or } T \overset{s}{\Longrightarrow} U,$$

from which it follows immediately that

$$(s, X) \in \text{failures}(S \,\Box\, T) \;\Leftrightarrow\; (s, X) \in \text{failures}(S) \cup \text{failures}(T).$$

Hence, for $s \neq \langle\rangle$,

$$\begin{aligned}(s, X) \in \text{failures}(S \,\Box\, T) &\Leftrightarrow (s, X) \in \mathcal{F}[\![P]\!] \cup \mathcal{F}[\![Q]\!]\\ &\Leftrightarrow (s, X) \in \mathcal{F}[\![P \,\Box\, Q]\!].\end{aligned}$$

That completes the proof. ∎

COROLLARY 1.4.4. *Two processes have the same failure sets if and only if their images under $\mathcal{T}$ are equivalent:*

$$\mathcal{F}[\![P]\!] = \mathcal{F}[\![Q]\!] \quad\Leftrightarrow\quad \mathcal{T}[\![P]\!] \equiv \mathcal{T}[\![Q]\!].$$

This result shows that the failure equivalence relation partitions the set of synchronisation trees into a set of equivalence classes which correspond precisely to the meanings of CSP processes under the failure set semantic function. Moreover, the preorder $\sqsubseteq$ on trees corresponds exactly to the nondeterminism order on processes. It is also clear that failure equivalence on trees respects the tree operations which represent CSP operations. Thus, unlike Milner's system, here we have an equivalence relation which is a *congruence* with respect to the operations of our language. Of course, it is still not the case that the equivalence relation is well behaved with respect to $+$. Again, $S \equiv \tau S$ always holds, but it is not in general true that $S + T \equiv \tau S + T$. But this does not matter to us, since $+$ is not a CSP operation.

2. Conclusions.

We have described two alternative languages for concurrency, CCS and CSP, using a common basis for a semantic model: Milner's synchronisation trees. We gave a semantics to CSP by mapping terms in this language to synchronisation trees and factoring out by an equivalence relation. Milner's semantics for CCS was also constructed in a similar fashion, although his equivalence relation was chosen in accordance with different criteria. We established a simple relationship between these two equivalence relations, and defined a complete axiom system for proving equivalence of finite CSP terms. Since the CCS equivalence is finer than the failure equivalence of CSP, and therefore makes fewer identifications on trees, all of the axioms of CCS are true also of failure equivalence. The converse is not true, of course, and the axioms of failure equivalence reflect very clearly the reasons.

Interesting topics for future work include extending these ideas to cope with infinite processes and infinite synchronisation trees; such an extension would be necessary if a form of recursion were added to the syntax of the language of processes. It then becomes necessary to treat problems associated with *divergence,* and there are several alternative models here to consider: for example, see [3,4,6] for CCS and [7,8] for CSP. To illustrate some of the problems introduced by divergence, consider the following. If we identify divergence with the ability to perform arbitrarily long sequences of hidden actions, without ever interacting with the environment, then one can model divergence as the presence in a synchronisation tree of an infinite path of τ arcs. Under the standard observational equivalence and failure equivalence presented here, the divergent tree τ^ω would be identified with *NIL*. Intuitively, such an inability to distinguish between divergence and deadlock is unappealing. One would therefore have to change the definitions of equivalence of trees to allow for such distinctions to be made. It is not yet clear which methods of modelling divergence are likely to be most successful.

An interesting area for research is to investigate the possibility of axiomatising other semantic models for concurrency, in much the same way as was done here for the failures model of CSP. A concise, complete proof system summarises the essential properties of a semantic model in a way that might clarify the differences and similarities between various models. Darondeau [9] gives an axiomatization of an equivalence on processes; these axioms can be simply derived from (B1)-(B4), showing that the same underlying model is being used, even though the constructions differ. In [6] the authors give complete proof systems for a variety of preorders (and hence for the associated equivalences) on CCS, which are shown in [7] to be related closely to the failures model. [7] also shows the connections between failure sets and the model proposed by Kennaway for communicating processes. It should be possible to axiomatize Kennaway's model, since it is so closely related to failure sets. Similarly, the *possible futures* model of communicating processes, described in [10], has connections with the failures model; an axiomatization for this model would again delineate precisely the differences and similarities.

3. Appendix.

In this section we prove the results stated earlier about normal forms, and justify our claim that the CSP axiom system is complete for finite trees.

First, let us recall that the axiom system consists of the following set of axioms and rules:

$$\begin{array}{ll}
\text{(B1)} & S + \tau T + U \equiv \tau(S+T) + \tau T + U \\
\text{(B2)} & \tau S \equiv S \\
\text{(B3)} & \mu S + \mu T + U \equiv \mu(\tau S + \tau T) + U \\
\text{(B4)} & \tau(\mu S + T) + \tau(\mu S' + T') \equiv \tau(\mu S + \mu S' + T) + \tau(\mu S + \mu S' + T')
\end{array}$$

$$\text{(R)} \qquad \frac{S \equiv S'}{\mu S + T \equiv \mu S' + T}.$$

First it will be useful to derive some generalisations of these axioms.

PROPOSITION A.1. *The following laws are derivable in our proof system:*

$$\text{(G1)} \qquad S + \sum_{i=1}^{n} \tau T_i + U \equiv \sum_{i=1}^{n} \tau(S + T_i) + \sum_{i=1}^{n} \tau T_i + U$$

$$\text{(G3)} \qquad \sum_{i=1}^{n} \mu S_i + U \equiv \mu \sum_{i=1}^{n} \tau S_i + U$$

$$\text{(G4)} \qquad \sum_{i=1}^{n} \tau(\mu S_i + T_i) \equiv \sum_{i=1}^{n} \tau(\mu S + T_i),$$

where $S = \sum_{i=1}^{n} \tau S_i$.

Proof. In each case an induction on n will establish the result; the base case, $n = 1$, is an instance of an axiom. Details are left to the reader. ∎

PROPOSITION A.2. *The following inference rule is a derived rule in our system:*

$$\frac{\sum_{i=1}^{n} \tau S_i \equiv \sum_{j=1}^{m} \tau T_j}{\sum_{i=1}^{n} \tau S_i + U \equiv \sum_{j=1}^{m} \tau T_j + U.}$$

Proof. Let us introduce abbreviations

$$S = \sum_{i=1}^{n} \tau S_i,$$
$$T = \sum_{j=1}^{m} \tau T_j.$$

Assume that $S \equiv T$ is provable. Then by (G3) with $\mu = \tau$, we can prove

$$(1) \quad \tau S + U \equiv S + U,$$
$$(2) \quad \tau T + U \equiv T + U.$$

But from $S \equiv T$ applying rule (R) gives us

$$(3) \qquad \tau S + U \equiv \tau T + U.$$

The result follows from (1) and (2). ∎

PROPOSITION A.3. *The following convexity laws are derivable:*

$$\text{(C1)} \qquad \tau S + \tau T \equiv \tau S + \tau T + \tau(S + T)$$
$$\text{(C2)} \qquad \tau S + \tau(S + T + U) \equiv \tau S + \tau(S + T) + \tau(S + T + U)$$

DEFINITION A.4. A tree T is *uniform* iff at each node in T if any (outgoing) arc is labelled τ then all are.

Thus, a tree T is uniform iff it has one of the two forms:

$$T = \sum_{i=1}^{n} \tau T_i,$$

$$T = \sum_{i=1}^{n} a_i T_i, \quad \text{where each } a_i \text{ is in } \Sigma, \text{ and each } T_i \text{ is also uniform.}$$

PROPOSITION A.5. *Any finite tree T is provably equivalent to a uniform tree.*

Proof. We use the following law, an instance of (G1):

$$S + \sum_{j=1}^{m} \tau T_j \equiv \sum_{j=1}^{m} \tau(S + T_j) + \sum_{j=1}^{m} \tau T_j.$$

Let T be the tree

$$T = \sum_{i=1}^{n} a_i S_i + \sum_{j=1}^{m} \tau T_j,$$

where the a_i are all in Σ. The proof is by induction on the depth of T. When T has depth zero it is trivial, because $T = NIL$ and this tree is already uniform. For the inductive step, assume that every tree of smaller depth than T is equivalent to a uniform tree. In particular, this means that there are uniform trees S_i', T_j' such that the relations

$$S_i \equiv S_i',$$
$$T_j \equiv T_j',$$

are provable. Using rule (R) this gives us

$$T \equiv \sum_{i=1}^{n} a_i S_i' + \sum_{j=1}^{m} \tau T_j'.$$

If any T_j' has τ branches at its root, we can use (G3) to contract successive τ labels; hence, we may assume without loss of generality that each T_j' has visible labels at its root. Writing S for the first term above, we have

$$\begin{aligned} T &= S + \sum_{j=1}^{m} \tau T_j' \\ &\equiv \sum_{j=1}^{m} \tau(S + T_j') + \sum_{j=1}^{m} \tau T_j' \qquad \text{by (G1).} \end{aligned}$$

But S and each T_j' are uniform, and they all have visible labels at the root; this means that each $(S + T_j')$ is also uniform, so the result follows. ▮

DEFINITION A.6. A set $\mathcal{B}$ of subsets of Σ is *convex* iff $\mathcal{B}$ is non-empty and for all $X, Z \in \mathcal{B}$,

$$X \cup Z \in \mathcal{B}$$
$$X \subseteq Y \subseteq Z \Rightarrow Y \in \mathcal{B}$$

For example, the set $\{\{a\},\{b\}\}$ is not convex, because it does not contain $\{a, b\}$. And the set $\{\emptyset, \{a,b\}\}$ is not convex because it does not contain $\{a\}$ and $\{b\}$.

DEFINITION A.7. A tree T is in *normal form* iff T has the structure

$$T = \sum_{B \in \mathcal{B}} \tau T_B,$$

with $\mathcal{B}$ convex, and each T_B having the form

$$T_B = \sum_{b \in B} bT_b,$$

where each T_b is also in normal form.

Notice that in a normal form, for each sequence of visible actions there is (at most) one derivative. For example, the tree

$$T = \tau aNIL + \tau(a\tau bNIL + bNIL)$$

is not in normal form, because there are two distinct a-derivatives. It can, however, be transformed by the axioms to the normal form

$$\tau aS + \tau(aS + bNIL),$$

where $S = \tau NIL + \tau bNIL$.

PROPOSITION A.8. *Any uniform tree is provably equivalent to a normal form.*

Proof. Any multiple occurrences of a visible label at a node can be combined into a single occurrence by the law (G3):

$$\sum_{i=1}^{n} aS_i + T \equiv a(\sum_{i=1}^{n} \tau S_i) + T.$$

Note that this transformation preserves uniformity. Sequences of τ arcs can always be pruned down to a set of single τ arcs by the same law:

$$\sum_{i=1}^{n} \tau S_i + T \equiv \tau(\sum_{i=1}^{n} \tau S_i) + T.$$

We can use the convexity laws (C1) and (C2), to introduce convexity at each node of the tree; finally, one uses (G4) to produce a unique s-derivative for each trace s of the tree. ▮

Note that a normal form is simply a tree T with structure:

$$T = \sum_{B \in \mathcal{B}} \tau \sum_{b \in B} bT_b,$$

where $\mathcal{B}$ is convex and each T_b is in normal form.

The completeness theorem rests upon the fact that equivalence on normal forms is provable. Indeed, it turns out that two normal forms are failure equivalent iff they are identical trees, up to order of summation. The proof will rely on a lemma.

PROPOSITION A.9. *For trees S and T in normal form, say*

$$S = \sum_{B \in \mathcal{B}} \tau \sum_{b \in B} bS_b,$$
$$T = \sum_{C \in \mathcal{C}} \tau \sum_{c \in C} cT_c,$$

$S \sqsubseteq T$ holds iff $\mathcal{C} \subseteq \mathcal{B}$, and for all $c \in C \in \mathcal{C}$, $S_c \sqsubseteq T_c$.

Proof.

Using the above notation, suppose $S \sqsubseteq T$. Then failures$(S) \supseteq$ failures(T). This immediately gives

$$\text{initials}(S) = \bigcup \mathcal{B} \supseteq \bigcup \mathcal{C} = \text{initials}(T).$$

If $\mathcal{C} \not\subseteq \mathcal{B}$, let C be a set in $\mathcal{C}$ but not in $\mathcal{B}$. Let $X = \Sigma - C$, so that $(\langle\rangle, X)$ is a failure of T. By hypothesis, this is also a failure of S, so there must be a $B \in \mathcal{B}$ such that $B \cap X = \emptyset$; equivalently, $B \subseteq C$. But then we have

$$B \subseteq C \subseteq \bigcup \mathcal{C} \subseteq \bigcup \mathcal{B},$$

and we know that B and $\bigcup \mathcal{B}$ belong to the convex set $\mathcal{B}$. Hence, $C \in \mathcal{B}$, which contradicts our assumption. It follows that $\mathcal{C}$ is a subset of $\mathcal{B}$.

Since S and T are in normal form, they have unique derivatives for each initial event. We have shown that initials$(S) \supseteq$ initials(T). It follows easily from the assumption that $S \sqsubseteq T$ that for all $a \in$ initials(T), $S_a \sqsubseteq T_a$.

To complete the proof we have to reverse the argument to obtain the converse implication. The details are simple, and left to the reader. ∎

COROLLARY A.10. *Two normal forms S and T as above are equivalent iff $\mathcal{B} = \mathcal{C}$ and for every $a \in \mathcal{B}$ the trees S_a and T_a are equivalent.*

PROPOSITION A.11. *(Completeness) Two finite trees S and T are failure equivalent iff $S \equiv T$ is provable in the above proof system.*

Proof. By Propositions A.5 and A.8 S and T are provably equivalent to normal forms, say S^* and T^*. Because the proof system is *sound,* we can assume that S^* and T^* are failure equivalent.

We will prove by induction on the depth of the trees that equivalence of normal forms is provable. The basis is trivial, because the only normal form of depth 0 is *NIL*. Let the two normal forms be

$$S^* = \sum_{B \in \mathcal{B}} \tau \sum_{b \in B} bS_b{}^*,$$
$$T^* = \sum_{C \in \mathcal{C}} \tau \sum_{c \in C} cT_c{}^*.$$

By Corollary A.10 we have $\mathcal{B} = \mathcal{C}$ and $S_a{}^* \equiv T_a{}^*$, for all a. Since $S_a{}^*$ and $T_a{}^*$ are also in normal form, and have smaller depth than S and T, we may assume by the inductive hypothesis that $S_a{}^* \equiv T_a{}^*$ is provable, for all a. An application of rule (R) then proves $S \equiv T$. ∎

4. Acknowledgements.

The author has benefitted from discussions with C.A.R.Hoare, Robin Milner, Bill Roscoe, Bill Rounds and Glynn Winskel. This work was supported in part by a grant from the Science Research Council of Great Britain, and by Carnegie-Mellon University Computer Science Department.

5. References.

[1] Milner, R., A Calculus for Communicating Systems, Springer LNCS Vol. 92 (1980).

[2] Hoare, C.A.R., Brookes, S.D., and Roscoe, A.W., A Theory of Communicating Sequential Processes, Technical Report PRG-16, Oxford University Computing Laboratory, Programming Research Group (1981).

[3] Hennessy, M.C.B. and Plotkin, G.D., A Term Model for CCS, Proceedings of 9th MFCS Conference, Springer LNCS Vol. 88 (1980).

[4] Hennessy, M.C.B. and Milner, R., On observing nondeterminism and concurrency, in: Springer LNCS Vol. 85 (1979).

[5] Hoare, C.A.R., Communicating Sequential Processes, CACM 21, Vol. 8 (1978).

[6] Hennessy, M., and de Nicola, R., Testing equivalences for processes, Technical Report, University of Edinburgh (July 1982).

[7] Brookes, S.D., A Model for Communicating Sequential Processes, Ph.D thesis, University of Oxford (submitted 1983).

[8] Roscoe, A.W., A Mathematical Theory of Communicating Sequential Processes, Ph.D thesis, University of Oxford (1982).

[9] Darondeau, Ph., An enlarged definition and complete axiomatization of observational congruence of finite processes, Proceedings of International Symposium on Programming, Springer LNCS 137 (1982).

[10] Rounds, W.C., and Brookes, S.D., Possible futures, acceptances, refusals, and communicating processes, Proceedings of 22nd IEEE Symposium on Foundations of Computer Science (October 1981).