Electronic Notes in Theoretical Computer Science 20 (2008)
URL: http://www.elsevier.nl/locate/entcs/volume20.html 20 pages

# Reasoning about recursive processes: Expansion is not always fair.

Stephen Brookes

*Department of Computer Science*
*Carnegie Mellon University*
*Pittsburgh, PA 15213, USA*

**Abstract**

When reasoning about parallel programs we would like to combine fixed-point laws for unrolling recursion with expansion laws for parallel composition. Algebraic manipulation using this combination is dangerous, because of the need to stay faithful to the assumption that parallel processes are executed fairly. We give an example of a finite-state parallel system in which the combination of fixed-point laws with a Milner-style expansion law causes a mismatch with fair parallel composition. Similar difficulties are well known in the literature, and it is traditional to conclude that the problem is caused by the unbounded non-determinism and inherent lack of continuity associated with fairness. Rather than laying the blame on fairness, we propose a new form of *fair expansion* for processes. Every finite-state process has a fair expansion, which characterizes its behavior when run for an arbitrary finite number of steps. We show that fair expansion interacts smoothly with recursion and with parallel composition. We provide a fair expansion rule for obtaining a valid expansion for a parallel system from fair expansions for its components. We establish a theorem on the recursive characterization of parallel compositions of recursive processes, in what amounts to a fair parallel generalization of Bekic's theorem concerning simultaneous recursive definitions. We also discuss how to incorporate local variables.

## 1 Introduction

When reasoning about safety and liveness properties of parallel networks it is vital to use a semantic model that assumes (weakly) fair execution, building in the guarantee that every process that has not terminated will eventually be scheduled for activity. This form of fairness is a convenient abstraction from network implementation details; reasonable schedulers using simple strategies such as "round-robin" do provide such guarantees. The assumption of fairness permits us to ignore irrelevant, unavailable, and imponderable book-keeping information about scheduling order. Moreover, many liveness properties do not hold unless fairness is assumed. For example, the fact that the Alternating Bit protocol properly transmits data from sender to receiver, despite the lossiness of the transmission medium, depends crucially on fair interaction [12].

Reasoning about parallel systems tends to be complicated because of the potentially complex patterns of interaction between processes running concurrently. A natural way to simplify the task and help to avoid a combinatorial explosion is to use algebraic laws of program equivalence. Such laws can streamline proofs of correctness, and can be applied without the need for detailed analysis of the actual semantics of a network. For example, Kahn's elegant and simple stream-based semantics for deterministic dataflow networks relies on elementary fixed-point theorems due to Scott, Bekic, and Vuillemin [9,5]. Fairness also plays a crucial role here: Kahn's semantics is only sound if the processes in a network are assumed to be executed fairly [5].

Several specification languages and calculi have been proposed for reasoning about concurrent systems, including Milner's CCS [12] and Hoare's CSP [7]. The semantics of CCS involves *synchronization trees* and employs (strong or weak) *bisimulation* as the notion of program equivalence [12]. The traditional semantics for CSP is based on *failure sets* and *divergence traces* [8]. Milner introduced an *expansion theorem* for reducing parallel compositions of CCS processes to a simpler form involving non-deterministic choice. Expansion laws have also been used by Roscoe *et al* as the basis for the development of the FDR model checker for finite-state CSP programs [6,16,1]. In each of these cases there is a variety of useful algebraic laws of process equivalence, sound with respect to the relevant semantic model. However, in neither of these cases was fairness built into the semantic framework, nor do the process equivalence laws accurately reflect fair execution.

As we have argued above, fairness is a vital assumption and we need a methodology for reasoning about processes that incorporates fairness. We would like to be able to combine laws concerning recursive process definition with laws specific to fair parallelism and be assured that "everything works out". By analogy, when reasoning about Kahn-style networks we can expand or contract recursive definitions for the individual nodes of a network and be sure that the semantics of the overall network remains unchanged: the fixed point obtained as the semantics of an "expanded" network coincides with the fixed point of the equations describing the original network. We would like to obtain similar guarantees in a more general setting. However, naïve use of expansion and recursion laws is dangerous, because of the need to remain faithful to the underlying fairness assumption. To put it crudely, expansion isn't fair. We give an example of a finite-state parallel system in which the combination of fixed-point laws, a Milner-style expansion law, and fair parallel composition leads to operationally misleading conclusions. Specifically, the result is an expansion which holds equally well in an unfair semantics and in a fair semantics, so that it fails to capture the process's intended behavior.

It has been traditional to point to the unbounded non-determinism and inherent lack of continuity engendered by fairness as the cause of these difficulties. Thus, it has been argued, fairness does not fit conveniently into standard fixed-point theory, so this kind of mismatch between fairness and recursion is only to be expected. Our response to this mismatch is different, since we regard fairness as a fundamental concept which must be incorporated smoothly in the semantics of parallel programs. Of course we also wish to retain the traditional treatment of recursion. Rather than laying the blame on fairness, we propose a new form of *fair expansion* for processes.

Whereas a Milner-style expansion reveals the initial atomic actions of a process, we use more general "prefix expansions" which specify a finite initial portion of a process's behavior. Prefix expansions are "closed under expansion", in the sense that we may freely perform expansions in any context without affecting the semantics of the system being analyzed. A *fair* expansion is a prefix expansion that reveals the process's potential behavior when run for an arbitrary finite number of steps. Every finite-state process has a fair expansion so that such expansions provide a widely applicable specification framework. We show that fair expansion interacts smoothly with recursion and with fair parallel composition. We provide a rule for generating a valid expansion for a network from fair expansions of its components. We state a theorem on the recursive characterization of parallel compositions of recursive processes, in what amounts to a fair parallel generalization of Bekic's theorem concerning simultaneous recursive definitions. We also show how a localized form of expansion can be obtained for a process which has local or private variables. This leads to a theorem concerning the solution to recursive process definitions in which each process is equipped with local variables. We illustrate the utility of our ideas and results by examining some simple examples.

## 2 Syntax and semantics

We use a CSP-like syntax for processes [7,4,5]; such notation was also used by Kahn [9,10] for deterministic dataflow networks, and we generalize to allow non-determinism. Unlike Hoare's CSP [7], but as in Kahn networks, we assume that communication is *asynchronous*: a process performing output never blocks, and a process attempting an input waits if the relevant channel is empty. Also unlike CSP, processes may share state. We assume *(weak) fairness*: in a parallel composition, each non-terminated process will eventually be scheduled.

Our semantic model uses "transition traces" [13,2–5]. We provide here a brief summary of the main technical details and concepts, which should provide sufficient background for the rest of the paper [1].

[1] In fact we begin with a simpler, less abstract form of trace semantics than was used in [2–5], working with traces in which each step represents a single atomic action. This permits a more straightforward account of recursion. Later we will show how to adapt our

A finite transition trace of a process $P$ has the form

$$(s_0, s_0')(s_1, s_1') \ldots (s_n, s_n')$$

and represents a computation of $P$ starting in state $s_0$, ending in $s_n'$, during which the process's "environment" makes a sequence of interruptions. The environment represents the rest of the network, i.e. the effect of other processes executing concurrently. Each step $(s_i, s_i')$ represents the effect of an atomic action performed by $P$, and each interference step from $s_{i-1}'$ to $s_i$ represents the effect of a finite sequence of atomic actions by the environment. An infinite transition trace of form

$$(s_0, s_0')(s_1, s_1') \ldots (s_n, s_n') \ldots$$

represents an infinite interactive computation of the same nature, assuming fair interaction between the process and environment.

Trace semantics can be defined denotationally[13,2–5]. Each program construct determines a *monotone* function on the complete lattice of trace sets (ordered by set inclusion). The traces of $P \| Q$ are *fair merges* of traces of $P$ and $Q$. The traces of $P; Q$ are obtained by concatenation. The traces of a recursively defined process are obtained as the *greatest fixed point* of the corresponding function on trace sets [17,2,3], whereas taking the *least fixed point* yields only the *finite* traces of the recursive process. We will write $tr(P)$ for the trace set of $P$.

Note our use of a single unifying semantic framework for shared-variable programs and communicating processes. We incorporate channels directly into the state, so that an input or output action is modelled as a particular kind of state change. The potential for deadlock is modelled by a form of infinite stuttering. We avoid the traditional reliance on channel names in the internal structure of traces, in contrast with Hoare-style traces. Although our semantics is "only" trace-theoretic it still permits reasoning about deadlock.

Trace semantics supports compositional analysis of safety and liveness properties. Thus when two processes have the same trace sets one may be substituted for the other in any network without affecting the correctness of the overall network.

## 3 Expansion and recursion

We now turn to finite-state processes. The results here may prove relevant to model-checking for finite-state networks, although we do not explore the potential for implementation of these ideas in this paper.

Using a Milner-style CCS-like notation, a finite-state process can be presented as a "summation" including a "branch" for each atomic action initially

ideas to the fully abstract trace semantics of [2], in which each step represents an arbitrary finite sequence of atomic actions.

possible for the process, leading to a representation of the process's behavior after this action.

**Definition 3.1 (Initial expansion)**
An *initial expansion* for process $P$ has the form

$$P = \sum_{i=1}^{n} a_i; P_i,$$

where each $a_i$ is an "atomic" expression representing a single atomic action such as an input or output, an expression evaluation, or an assignment, and each $P_i$ is a process. For a finite-state process $P$ each of the processes $P_i$ will also be finite-state.

We use the term *initial expansion*, or *Milner-style expansion*, rather than simply "expansion", to avoid confusion with what follows.

Although originally phrased in terms of CCS *synchronization trees* [12], with equality interpreted as *bisimulation* [14], this kind of initial expansion formula can also be used in the trace-theoretic setting, with equality interpreted as *trace equivalence*. We say that the expansion formula $P = \sum_{i=1}^{n} a_i; P_i$ is *sound* (or *valid*) if the traces of $P$ are accurately represented as the union of the traces of the branches, i.e. if

$$tr(P) = \bigcup_{i=1}^{n} \{\alpha_i\beta_i \mid \alpha_i \in tr(a_i) \ \& \ \beta_i \in tr(P_i)\}.$$

This kind of expansion generalizes to mutually recursive process definitions as follows. The notation $\mathbf{rec}\,(p_1, \ldots p_k).\,(Q_1, \ldots, Q_k)$ represents a tuple of $k$ processes, defined by mutual recursion; each of the process variables $p_1, \ldots, p_k$ may appear in $Q_1, \ldots, Q_k$. An expansion for such a term has the form

$$\mathbf{rec}\,(p_1, \ldots, p_k).\,(Q_1, \ldots, Q_k) = \mathbf{rec}\,(p_1, \ldots, p_k).\,(\sum_{i_1=1}^{n_1} a_{i_1}; P_{i_1}, \ldots, \sum_{i_k=1}^{n_k} a_{i_k}; P_{i_k})$$

where we allow $p_1, \ldots, p_k$ to occur free in the $P_{i_j}$. The soundness criterion for such an expansion is the obvious generalization of the above.

It is convenient to present recursive definitions via sets of equations. For example the equations

$$p = a!0; p + b!0; q \qquad q = c!1; q$$

correspond to the term $\mathbf{rec}\,(p, q).\,(a!0; p + b!0; q, c!1; q)$. Taking the greatest fixed-point of the corresponding functional yields the trace sets

$$tr(p) = (a!0)^*; b!0; (c!1)^\omega \ + \ (a!0)^\omega \qquad tr(q) = (c!1)^\omega,$$

where we take the liberty of writing $a!0$ (and so on) to denote the corresponding trace set $tr(a!0)$. This is consistent with the above expansion, since the trace sets given by these formulae do satisfy the intended equations:

$$tr(p) = a!0; tr(p) + b!0; tr(q) \qquad tr(q) = c!1; tr(q).$$

The standard *fixed-point theorem* for reasoning about recursive definitions asserts that

$$\mathbf{rec}\ p.P = P[(\mathbf{rec}\ p.P)/p].$$

This law is valid in our semantics, since

$$tr(\mathbf{rec}\ p.P) = tr(P[(\mathbf{rec}\ p.P)/p]).$$

Similarly, *greatest fixed-point induction* is a valid rule of inference: whenever $P = F(P)$ it follows that $P \subseteq \mathbf{rec}\ p.\, F(p)$.

Milner introduced an *expansion law* for “reducing” parallel composition to non-deterministic choice [12], producing an initial expansion for $P\|Q$ from initial expansions for $P$ and $Q$. Milner’s original law was designed to model a synchronizing form of parallel composition, and was sound with respect to bisimulation. We deal here with asynchronous processes and trace equivalence. The appropriate *asynchronous expansion law* for our setting is the following.

**Theorem 3.2 (Asynchronous expansion law)**
*If $P = \sum_{i=1}^{n} a_i; P_i$ and $Q = \sum_{j=1}^{m} b_j; Q_j$ are valid, then*

$$P\|Q = \sum_{i=1}^{n} a_i; (P_i\|Q) \ + \ \sum_{j=1}^{m} b_j; (P\|Q_j)$$

*is valid.*

This expansion law is sound in our fair semantics. If the traces of $P$ and $Q$ are accurately characterized by the given expansions for $P$ and $Q$, then the traces of $P\|Q$ coincide with the traces of the expanded form given here. The assumption that the $a_i$ and $b_j$ represent *atomic* actions is crucial: if the $a_i$ had non-trivial computations (and therefore had non-trivial traces of length greater than 1) the rule would cease to be sound.

## 4 Expansion isn’t always fair

As we have remarked, the Milner-style asynchronous expansion law is a sound rule of inference when parallel composition is interpreted as fair merge of trace sets. It is tempting to believe that the presence of summands coming from $P$ as well as summands coming from $Q$ on the right-hand side “builds in” fairness. However, it is easy to see that this is not the case: the above expansion rule is still sound if we interpret parallel composition *unfairly* on both sides of the equation. In this sense, the Milner-style expansion law does not encapsulate fairness.

As a result of this insensitivity to the distinction between fair and unfair execution, we need to be careful when combining recursion and expansion.

This is illustrated by the following example. Let $p$ and $q$ be defined by

$$p = a!0; p \qquad q = a!1; q.$$

Thus $tr(p) = (a!0)^\omega$ and $tr(q) = (a!1)^\omega$. The above equations amount to an initial expansion for these processes. Since we assume fair execution, $tr(p\|q)$ consists of the traces containing output of an infinite number of 0's and an infinite number of 1's on channel $a$. Using the fixed-point property (in the forwards direction), the asynchronous expansion law, and the fixed-point property again (backwards), we can derive the following initial expansion for $p\|q$ from the above equations:

$$\begin{aligned} p\|q &= (a!0; p)\|(a!1; q) \\ &= a!0; (p\|(a!1; q)) \ + \ a!1; ((a!0; p)\|q) \\ &= a!0; (p\|q) \ + \ a!1; (p\|q). \end{aligned}$$

Since each step in this derivation is semantically sound, so is the conclusion: $p\|q$ is a fixed-point of the corresponding functional. Indeed, every trace of $p\|q$ does begin either with $a!0$ or with $a!1$, and the subsequent behavior is the same as that of $p\|q$. However, the corresponding recursion $r = a!0; r + a!1; r$ does not have a unique fixed-point. Its *greatest* fixed-point is $(a!0 + a!1)^\omega$, which includes traces such as $(a!0)^\omega$ that do not represent fair executions of $p\|q$. In fact this greatest fixed-point really represents the *unfair* parallel composition of $p$ and $q$. The *least* fixed-point is the empty set, which also fails to represent the intended behavior. Thus the expansion equation does not uniquely characterize the intended trace set.

This simple example is symptomatic of a pervasive problem: we find that a process satisfies a recursive equation and we seek assurance that this fully and uniquely characterizes the process. This kind of problem arises frequently when trying to show that a process $p$ defined as the greatest fixed-point of a functional $F$ coincides with the process or specification $q$ defined as the greatest fixed-point of a functional $G$. Showing that $p = G(p)$ is insufficient to allow the conclusion that $p = q$; it may be difficult to show the "converse", i.e. that $q = F(q)$; and the functions $F$ and $G$ may not have unique fixed-points.

There is a problem, then, with the interaction between fair parallelism, recursive definitions, and Milner-style expansions[2]. Since the *fairmerge* relation on trace sets can be characterized as a greatest fixed-point [2], so that one might reasonably expect fair parallelism to interact smoothly with recursion, we will focus our attention on the form of expansion chosen for processes.

The problem stems from the decision to use an expansion form based on

[2] Such problems are dealt with in a different manner in Parrow's work on fair process algebras [15]; using Milner-style expansion laws and bisimulation as the behavioral equivalence, Parrow incorporates fairness "on top of" an unfair semantics by means of an *infinitary restriction* operator.

initial actions. In contrast, the fairmerge relation is most naturally described in terms of letting each process run for an arbitrary finite number of steps at a time. Intuitively, a Milner-style expansion is too "shallow", and we propose instead a "deep" form of expansion. Again, the issue here is not *soundness*: Milner's expansion law, and the usual fixed-point laws, are all semantically sound. But we want to provide a form of expansion, and a methodology for reasoning about parallel composition, which accurately encapsulates fairness.

## 5 Fair expansion

We now propose a form of expansion in which the branches incorporate "steps" based on finite prefixes of traces, rather than initial atomic actions. Bearing in mind our desire to stay faithful to fair execution, we therefore make the following definition.

**Definition 5.1 (Fair expansion)**
A *fair expansion* for a process $P$ is a summation of the form

$$P = \sum_{i=1}^{n} A_i; P_i$$

where each $A_i$ denotes a set of (non-empty) finite traces, and the following conditions hold:

(i) $tr(P) = \bigcup_{i=1}^{n}\{\alpha_i\beta_i \mid \alpha_i \in tr(A_i) \ \& \ \beta_i \in tr(P_i)\}$

(ii) $\forall \alpha \in tr(P).\ \forall \beta \leq_{fin} \alpha.\ \exists i.\ \beta \in tr(A_i) \ \& \ \alpha\,\mathbf{after}\,\beta \in tr(P_i)$.

The first condition says that the traces are accurately described by the expanded form. The second says that every finite prefix $\beta$ of a trace of $P$ belongs to one of the $A_i$, and its suffix belongs to the corresponding $P_i$. (When $\beta \leq \alpha$ we write $\alpha\,\mathbf{after}\,\beta$ to denote this suffix.) Intuitively, this means that the branches $A_i$ $(i = 1, \ldots, n)$ are "deep" enough to account for all possible finite partial runs of $P$.

As before, we generalize to mutually recursive process definitions: a formula of the form

$$\mathbf{rec}\,(p_1, \ldots, p_k).\,(Q_1, \ldots, Q_k) = \mathbf{rec}\,(p_1, \ldots, p_k).\,(\sum_{i_1=1}^{n_1} A_{i_1}; P_{i_1}, \ldots, \sum_{i_k=1}^{n_k} A_{i_k}; P_{ik}),$$

where we allow $p_1, \ldots, p_k$ to occur free in the $P_{i_j}$, is a fair expansion if the obvious generalized version of the above criteria hold, for the tuples of trace sets obtained by solving the corresponding fixed-point equations.

Returning again to the example, the formula

$$\mathbf{rec}\,(p, q).\,(a!0; p,\ a!1; q) = \mathbf{rec}\,(p, q).\,(a!0; a!0; p,\ a!1; a!1; q)$$

is *not* a fair expansion, since (for instance) the first component process has traces beginning with arbitrarily long sequences of outputs, and these prefixes are not all traces of $(a!0; a!0)$. Instead, the formula

$$\mathbf{rec}\,(p,q).\,(a!0;p,\ a!1;q) = \mathbf{rec}\,(p,q).\,((a!0)^+;p,\ (a!1)^+;q)$$

is a fair expansion, which may be written equivalently in equational form as

$$p = (a!0)^+;p \qquad q = (a!1)^+;q$$

Every non-empty finite partial run of the first process belongs to $(a!0)^+$, and similarly for the second process and $(a!1)^+$; thus the criteria for "deepness" of the expansion branches are met.

Every finite-state process has a fair expansion. Moreover, fair expansions for the parallel components of a network can be used to derive a valid expansion for the entire network, as shown by the following *fair expansion law.*

**Theorem 5.2 (Fair expansion law)**
*If $P = \sum_{i=1}^n A_i;P_i$ and $Q = \sum_{j=1}^m B_j;Q_j$ are fair expansions, then*

$$P\|Q = \sum_{i=1}^n \sum_{j=1}^m (A_i\|B_j);(P_i\|Q_j)$$

*is a valid expansion.*

**Proof** Assume that the expansion formulas given for $P$ and $Q$ are fair:

(i) $tr(P) = \bigcup_{i=1}^n tr(A_i); tr(P_i)$

(ii) $\forall\alpha \in tr(P).\ \forall\alpha' \leq_{fin} \alpha.\ \exists i.\ \alpha' \in tr(A_i)\ \&\ \alpha\,\mathbf{after}\,\alpha' \in tr(P_i)$

(iii) $tr(Q) = \bigcup_{j=1}^m tr(B_j); tr(Q_j)$

(iv) $\forall\beta \in tr(Q).\ \forall\beta' \leq_{fin} \beta.\ \exists j.\ \beta' \in tr(B_j)\ \&\ \beta\,\mathbf{after}\,\beta' \in tr(Q_j)$

We need to show that

- $tr(P\|Q) = \bigcup_{i=1}^n \bigcup_{j=1}^m tr(A_i\|B_j); tr(P_i\|Q_j)$

Inclusion from right to left is easy, i.e.

$$tr(P\|Q) \supseteq \bigcup_{i=1}^n \bigcup_{j=1}^m tr(A_i\|B_j); tr(P_i\|Q_j),$$

although it is worth commenting that this does rely on the built-in assumption that the $A_i$ and $B_j$ contain only non-empty and finite traces.

To show inclusion from left to right, suppose $\gamma \in tr(P)\|tr(Q)$. Thus $\gamma$ is a fair merge of a trace $\alpha \in tr(P)$ with a trace $\beta \in tr(Q)$. By (i) and (iii), there are $i, j$ such that $\alpha \in tr(A_i;P_i)$, $\beta \in tr(B_j;Q_j)$. By (ii), (iv), and the definition of fairmerge some sufficiently long prefix of $\gamma$ is a fair merge of a trace from $A_i$ with a trace from $B_j$, and the rest of $\gamma$ is a fair merge of a trace of $P_i$ with a trace of $P_j$. Thus $\gamma$ belongs to $tr(A_i\|B_j); tr(P_i\|Q_j)$, as required. □

Note the differences between the *fair expansion law* and the Milner-style law discussed earlier. Here the $A_i$ and $B_j$ denote (non-empty) sets of finite traces, and so do the $A_i\|B_j$, whereas in the earlier kind of expansion the $a_i$ represent atomic actions, which amount to trace sets containing only traces of length 1. Intuitively, our expansion law is sound because of the built-in property that the $A_i$ and $B_j$ are "sufficiently deep" to account for all finite prefixes of all traces of $P$ and $Q$, respectively.

Again revisiting the example, letting $P = \mathbf{rec}\, p.a!0; p$ and $Q = \mathbf{rec}\, q.a!1; q$, we have fair expansions

$$P = (a!0)^+; P$$

$$Q = (a!1)^+; Q,$$

so we can conclude that

$$P \| Q = (a!0)^+ \| (a!1)^+;\ (P \| Q)$$

is a valid expansion. Note that the greatest fixed-point of the corresponding functional is $((a!0)^+ \| (a!1)^+)^\omega$, and this is indeed the trace set for $(a!0)^\omega \| (a!1)^\omega$. Thus we obtain an expansion for $P \| Q$ which accurately captures the intended behavior and embodies fair parallel interaction: this expansion is not valid in an unfair semantics.

We do *not* propose fair expansions as *canonical*: a process will typically have many distinct fair expansions. For example, in addition to the fair expansion given above, $\mathbf{rec}\ p.\, a!0; p$ also has the fair expansion $p = (a!0)^+; (a!0)^+; p$. Nevertheless all fair expansions for a given process are *semantically* equivalent, since they all denote the same trace set. The differences concern the choice of $A_i$, and there is typically no canonical choice. It might be interesting to find syntactic constraints on the structure of an expansion sufficient to ensure that every process has a unique fair expansion obeying the constraints. Such a form of expansion might then serve as a normal form, so that to prove equivalence of two processes it would suffice to show that they have the same normal form. In contrast, Milner-style expansions are unique, modulo the order in which the summands are listed (and number of times each summand is included). However, the lack of canonicity can also be regarded as a virtue: unlike initial expansions, prefix expansions are *closed under expansion*, in the following sense.

**Theorem 5.3** *If* $P = \sum_{i=1}^{n} A_i; P_i$ *is a valid expansion and, for each* $i \in 1 \ldots n$, $P_i = \sum_{j=1}^{n_i} B_{i_j}; Q_{i_j}$ *is a valid expansion, so is*

$$P = \sum_{i=1}^{n} \sum_{j=1}^{n_i} (A_i; B_{i_j}); Q_{i_j}.$$

**Proof** Immediate by associativity of trace concatenation. □

Although this property might seem too trite to bother with, it does give the guarantee that we can freely substitute expanded forms into calculations involving processes without concern: whatever expansion we arrive at will still qualify as valid.

The fair expansion law shows how to derive a *valid* expansion for $P \| Q$ from *fair* expansions for $P$ and $Q$. Although the parallel composition of two fair expansions will not generally be *fair* itself, it will be "almost fair", in that every finite prefix of a trace of the process *extends* to a trace belonging to one of the $A_i \| B_j$ with suffix belonging to $P_i \| Q_j$. This is enough to guarantee that the expansion obtained for $P \| Q$ will characterize this process uniquely, in the sense that it captures the essence of fair parallel interaction between $P$ and $Q$. Obviously a fair expansion is also almost fair, but the converse may fail. The property of "almost fairness" is crucial in justifying what follows below concerning the interaction between recursion and parallel composition, but it is worth noting that the notion of almost fair expansion is strictly weaker than the notion of fair expansion. The problem is that parallel composition of almost fair expansions for $P$ and for $Q$ may produce an *invalid* expansion for $P \| Q$. To see this consider the following almost fair expansions for $P = \mathbf{rec}\, p.a!0; p$ and $Q = \mathbf{rec}\, q.a!1; q$:

$$P = (a!0; a!0)^+; P \qquad Q = (a!1; a!1)^+; Q.$$

These expansions are almost fair because every non-empty prefix of a trace of $P$ either is or extends by an additional step to a trace in $(a!0; a!0)^+$, and similarly for $Q$. Parallel composition according to the fair expansion law would produce the expansion

$$P \| Q = (a!0; a!0)^+ \| (a!1; a!1)^+; P \| Q,$$

but this is not a valid expansion, since $P \| Q$ has the trace

$$a!0; a!1(a!0; a!0; a!1; a!1)^\omega,$$

which has no non-trivial finite prefix belonging to $(a!0; a!0)^+ \| (a!1; a!1)^+$, since none of its non-empty prefixes contains an even number of 0's and an even number of 1's. It follows that this trace does not belong to the expanded process, and hence that the expansion is invalid.

To demonstrate the way fair expansion interacts with recursion we now establish a theorem concerning the fair parallel composition of recursively defined processes. We state here a specialized version, assuming that the processes are described according to a particular (and common) tail-recursive template.

**Theorem 5.4 (Parallel recursion law)**
*Suppose the following is a fair recursive expansion for the processes variables $p_1, \ldots, p_n$:*

$$\begin{aligned} p_1 &= A_{11}; p_1 \ + \ A_{12}; p_2 \ + \ \cdots \ + \ A_{1n}; p_n \\ p_2 &= A_{21}; p_1 \ + \ A_{22}; p_2 \ + \ \cdots \ + \ A_{2n}; p_n \\ \cdots & \\ p_n &= A_{n1}; p_1 \ + \ A_{n2}; p_2 \ + \ \cdots \ + \ A_{nn}; p_n. \end{aligned}$$

*Let $p_{ij}$ $(i, j = 1, \ldots, n)$ be the processes defined recursively by the $n^2$ equations*

$$p_{ij} \;=\; \sum_{i',j' \in 1\ldots n} (A_{ii'} \| A_{jj'}); p_{i'j'} \qquad (i, j = 1 \ldots n).$$

*Then for each $i$ and $j$ we have $p_i \| p_j = p_{ij}$.*

**Proof** Let $\top$ denote the tuple of length $n^2$ all of whose components are equal to the set of all possible traces; this is the maximal element of the lattice of trace sets. Let $F$ be the functional corresponding to the equations for the $p_{ij}$ It is easy to show that the greatest fixed-point of $F$ is the tuple whose components are given by

$$p_{ij} = \bigcap_{k=0}^{\infty} F^k(\top) \;=\; \bigcup_{i_1,i_2,\ldots} \bigcup_{j_1,j_2,\ldots} ((A_{ii_1} \| A_{jj_1}); (A_{i_1 i_2} \| A_{j_1 j_2}); \ldots),$$

where each $i_k$ and $j_k$ ranges over the index set $\{1, \ldots, n\}$. Similarly the processes $p_i$ have trace sets

$$p_i = \bigcup_{i_1,i_2,\ldots} (A_{1i_1}; A_{i_1 i_2}; \ldots).$$

It is easy to see from these characterizations that every trace of $p_{ij}$ is also a possible trace for $p_i \| p_j$. For the converse inclusion we argue as in the proof of the fair expansion law, this time to show that for all $k \geq 0$ we have

$$(p_1 \| p_1, \ldots, p_i \| p_j, \ldots, p_n \| p_n) \subseteq F^k(\top).$$

The base case holds trivially. For the inductive step, suppose that the above inclusion holds for $k = m$. Using the fair expansion law on the given fair expansions for the $p_i$ we deduce that

$$p_i \| p_j = \sum_{i',j' \in 1\ldots n} (A_{ii'} \| A_{jj'}); (p_{i'} \| p_{j'})$$

is a valid expansion, for each $i$ and $j$. By definition of $F$, monotonicity of $F$, and the induction hypothesis it follows that

$$\begin{aligned}(p_1 \| p_1, \ldots, p_i \| p_j, \ldots, p_n \| p_n) &= F(p_1 \| p_1, \ldots, p_i \| p_j, \ldots, p_n \| p_n) \\ &\subseteq F(F^m(top)) \\ &= F^{m+1}(\top),\end{aligned}$$

as required. $\square$

This theorem expresses a parallel composition of two fixed-points as a (component of) another fixed-point. This kind of result proves very helpful in analyzing network designs in which component processes are recursively defined. The theorem can be viewed as a fair parallel generalization of Bekic's Theorem on multiple fixed-points. Despite its apparent simplicity, this result is worth noting because it provides justification for algebraic reasoning based on expanding processes inside recursive network descriptions, since we obtain the guarantee that the overall network's behavior, characterized as a greatest fixed-point, is unaffected by such changes. Thus our theorem is also in the spirit of Kahn's work on dataflow networks.

The example discussed earlier is a special case of this theorem, and can be formulated as $(\mathbf{rec}\ p.\, a!0; p) \,\|\, (\mathbf{rec}\ q.\, a!1; q) \;=\; \mathbf{rec}\ r.\, ((a!0)^+ \| (a!1)^+);\, r$.

### *5.1 Regular expansions*

We call an expansion $P = \sum_{i=1}^{n} A_i; P_i$ *regular* if each of the $A_i$ is a *regular expression*[3] and each $P_i$ is again a regular fair expansion.

Similarly we refer to a set of expansion equations as *regular* if each of the $A_{i_j}$ is a regular expression and each of the $P_{i_j}$ is regular.

The range of applicability of this kind of expansion is suggested by the following result, whose proof is straightforward and relies on elementary formal language theory.

**Theorem 5.5** *Every finite-state process has a regular fair expansion.*

Note that the fair expansion law preserves regularity: if the expansions for $P$ and $Q$ are regular, so is the expansion obtained for $P\|Q$. This is because the class of regular languages is closed under *shuffle*, which coincides with fair merge. We can appeal to algebraic laws concerning regular languages in proving inclusions or equivalences between the terms $A_i \| B_j$ occurring in such expansions of different processes [11].

Similarly if we have compose regular expansions we obtain another regular expansion.

## 6 Dealing with local variables

Local variables can be used to delimit the scope of interaction between parallel processes, for example to model a channel of interaction shared by certain processes but inaccessible to others. The notation **local** $h$ **in** $P$ denotes a process $P$ equipped with a local variable $h$; processes outside this scope do not affect $h$. When the type of $h$ indicates that it is a communication channel, the traces of **local** $h$ **in** $P$ are obtained by projection (ignoring the $h$-component) from traces of $P$ in which initially $h$ is empty and the contents of $h$ are never altered across step boundaries, since the environment has no access to $h$. Similarly, when $x$ is an integer-valued local variable and $n$ is an integer, we write **local** $x = n$ **in** $P$ for the process whose traces are obtained by projection (ignoring the $x$-component) from traces of $P$ in which initially the value of $x$ is $n$ and the value of $x$ is never changed across step boundaries. We also find it useful to employ similar notation for local channels, writing **local** $h = \rho$ **in** $P$,

[3] A regular expression is built up from atomic actions like $h!v$ and $h?v$ using the usual operations of concatenation, iteration, and union. We also permit the use of parallel composition, and we write $A^+ = AA^*$. In *$\omega$-regular expressions* we use $A^\omega$ for infinite iteration.

where $\rho$ is a finite sequence of data assumed to be the initial contents of the channel; this coincides with the above definition when we take $\rho$ to be the empty sequence.

The ideas introduced earlier in this paper can be extended to incorporate local variable declarations. Here we will discuss briefly how to deal with systems in which local variables take on only a bounded number of values, so that each process is finite-state.

In order to provide a form of expansion theorem coping with local variables we first need to introduce some notation. When $A$ is a set of finite traces we say that a Hoare-style formula $\{x = v\}A\{x = v'\}$ is valid if every trace in $A$ which is interference-free for $x$ and starts from a state in which $x = v$ ends in a state satisfying $x = v'$. We write $\vdash \{x = v\}A\{x = v'\}$ to indicate validity. We also write **local** $x = v$ **in** $A$ to denote the traces obtained from $A$ by projecting out the local variable $x$. We then obtain the following *Local Expansion Theorem*:

**Theorem 6.1 (Local expansion law)**
*If* $P = \sum_{i=1}^{n} A_i; P_i$ *is a valid expansion and for each* $i$ *we have*

$$\vdash \{x = v\}A_i\{x = v_i\}$$

$$A_i' =_{def} \textbf{\textit{local}}\ x{=}v\ \textbf{\textit{in}}\ A_i,$$

*then*

$$\textbf{\textit{local}}\ x{=}v\ \textbf{\textit{in}}\ P \ = \ \sum_{i=1}^{n} A_i'; \textbf{\textit{local}}\ x{=}v_i\ \textbf{\textit{in}}\ P_i$$

*is valid.*

We also obtain the following *Local Recursion Theorem*, in which we assume that the $p_i$ and $q_i$ are distinct process variables.

**Theorem 6.2 (Local recursion theorem)**
*If*

$$p_i = \sum_{j=1}^{n_i} A_{ij}; p_j \quad (1 \leq i \leq n)$$

*is a valid recursive expansion and for each* $i$ *and* $j$ *we have*

$$\vdash \{x = v_i\}A_{ij}\{x = v_j\}$$

$$A_{ij}' =_{def} \textbf{\textit{local}}\ x = v_i\ \textbf{\textit{in}}\ A_{ij},$$

*then*

$$q_i = \sum_{j=1}^{n_i} A_{ij}'; q_j \quad (1 \leq i \leq n)$$

*is valid. The fixed-points satisfy* $q_i \ = \ \textbf{\textit{local}}\ x = v_i\ \textbf{\textit{in}}\ p_i$.

## 7 Example: a mutual exclusion protocol

As an example to illustrate our methodology, consider the following semaphore-based algorithm for achieving mutual exclusion:

$$\textbf{local}\ \mathit{sem} = \textbf{true in}\ (P_0 \| P_1),$$

where

$$\begin{aligned}
P_0 = {} & \textbf{while true do} \\
& \quad (\textbf{await}\ \mathit{sem}\ \textbf{then}\ \mathit{sem}{:=}\textbf{false}; \\
& \quad \mathit{crit}_0; \\
& \quad \mathit{sem}{:=}\textbf{true}; \\
& \quad \mathit{noncrit}_0) \\
P_1 = {} & \textbf{while true do} \\
& \quad (\textbf{await}\ \mathit{sem}\ \textbf{then}\ \mathit{sem}{:=}\textbf{false}; \\
& \quad \mathit{crit}_1; \\
& \quad \mathit{sem}{:=}\textbf{true}; \\
& \quad \mathit{noncrit}_1)
\end{aligned}$$

and where $\mathit{crit}_i$ and $\mathit{noncrit}_i$ represent critical and non-critical sections of code, respectively, and *sem* does not occur free in any of this code. The intention is to permit $P_0$ and $P_1$ to keep iterating through their loop bodies while preventing simultaneous occurrence of $\mathit{crit}_0$ and $\mathit{crit}_1$. It is easy to see intuitively that this mutual exclusion property is guaranteed by the way the two processes use the semaphore variable *sem*. However, the algorithm also permits starvation, since it is possible for one process to get stuck waiting forever for the semaphore to be released while the other process keeps running on and on. We now sketch how this intuition can be supported by formal analysis using expansions.

The two processes have fair expansions of the following shape:

$$\begin{aligned}
P_0 &= \mathit{cycle}_0; P_0 + \ldots + \mathit{wait}; P_0 \\
P_1 &= \mathit{cycle}_1; P_1 + \ldots + \mathit{wait}; P_1
\end{aligned}$$

Here $\mathit{cycle}_i$ represents $P_i$ going all the way through its loop body a non-zero number of times, and *wait* represents a non-zero number of idle steps in which the value of $s$ is **false**. The omitted terms involve cases where $P_i$ gets part way through an iteration of its loop body; we do not need to discuss these cases to demonstrate the potential for starvation. Let us write

$$\begin{aligned} grab &= \{(s, s') \in tr(sem{:=}\mathbf{false}) \mid s \vdash sem = \mathbf{true}\} \\ idle &= \{(s, s) \mid s \vdash sems = \mathbf{false}\} \\ rel &= tr(sem{:=}\mathbf{true}) \end{aligned}$$

so that $tr(\mathbf{await}\ sem\ \mathbf{then}\ sem{:=}\mathbf{false}) = idle^*; grab \cup idle^\omega$. Then

$$\begin{aligned} cycle_0 &= (idle^*; grab; crit_0; rel; noncrit_0)^+ \\ cycle_1 &= (idle^*; grab; crit_1; rel; noncrit_1)^+ \\ wait &= idle^+ \end{aligned}$$

We thus obtain a valid expansion for $P_0 \| P_1$ with the following shape:

$$\begin{aligned} P_0 \| P_1 = {} & (cycle_0 \| cycle_1); P_0 \| P_1 + \\ & (cycle_0 \| wait); P_0 \| P_1 + \\ & (wait \| cycle_1); P_0 \| P_1 + \\ & (wait \| wait); P_0 \| P_1 + \ \ldots \end{aligned}$$

When we focus on the traces in which *sem* is interference-free and is assumed to start with the value **true** we see that:

- $\mathbf{local}\ sem = \mathbf{true}\ \mathbf{in}\ (cycle_0 \| wait) \ = \ (crit_0; noncrit_0)^+ \| stut^+$
- $\vdash \{sem = \mathbf{true}\}(cycle_0 \ \| \ wait)\{sem = \mathbf{true}\}$

where *stut* is the set of trivial one-step "stutter" traces which make no change to the state. Hence, using the Local Expansion Theorem, there is a valid expansion with the following shape:

$$\begin{aligned} & \mathbf{local}\ sem = \mathbf{true}\ \mathbf{in}\ (P_0 \| P_1) \\ & \quad = \ (crit_0; noncrit_0)^+ \| stut^+; \mathbf{local}\ sem = \mathbf{true}\ \mathbf{in}\ (P_0 \| P_1) \\ & \quad + \ \ldots \end{aligned}$$

It thus follows that

$$\begin{aligned} \mathbf{local}\ s = \mathbf{true}\ \mathbf{in}\ (P_0 \| P_1) & \supseteq ((crit_0; noncrit_0)^+ \| stut^+)^\omega \\ & = (crit_0; noncrit_0)^\omega \| stut^\omega \end{aligned}$$

showing that it is possible for $P_1$ to suffer starvation. Of course a symmetric possibility also exists for $P_0$.

## 8 Imposing closure

So far we have worked with traces built from atomic steps. This version of trace semantics is closely correlated with a standard operational semantics:

every step in every trace can be shown to correspond with a transition justified operationally. However, as we argued in [2], this yields a model in which certain fundamental laws of process equivalence fail to hold. Notably, the processes $P; \mathbf{skip}$ and $\mathbf{skip}; P$ denote different trace sets from that of $P$, although it is reasonable to expect that these three processes should be indistinguishable. This problem can be solved by imposing two natural closure conditions – *stuttering* and *mumbling* – on trace sets, as described in [2]. This leads to a semantics in which a process denotes a *closed* set of traces, and each step in a trace represents the effect of a finite sequence of atomic actions performed by the process, rather than representing a single atomic action. Every step in such a trace corresponds with a finite sequence of operationally justified transitions, and this semantics validates the equivalence of $P$, $P; \mathbf{skip}$, and $\mathbf{skip}; P$. In addition this semantics validates a number of natural laws of equivalence which can be extremely useful in mitigating the analysis of parallel systems. We now sketch how to incorporate closure into the fair expansion framework.

Let $tr^\dagger(P)$ denote the closure of the trace set of $P$. The expansion laws and recursion theorems given above can be recast into this framework, using $tr^\dagger$ instead of $tr$ in the definitions and theorems. In particular, we can interpret validity of an expansion in terms of $tr^\dagger$, so that $P = \sum_{i=1}^{n} A_i; P_i$ is valid if

$$tr^\dagger(P) = \bigcup_{i=1}^{n} \{\alpha_i \beta_i \mid \alpha_i \in tr(A_i) \ \&\ \beta_i \in tr(P_i)\}^\dagger.$$

We then say that an expansion is fair if

(i) $tr^\dagger(P) = \bigcup_{i=1}^{n} \{\alpha_i \beta_i \mid \alpha_i \in tr(A_i) \ \&\ \beta_i \in tr(P_i)\}^\dagger$

(ii) $\forall \alpha \in tr^\dagger(P).\ \forall \beta \leq_{\mathit{fin}} \alpha.\ \exists i.\ \beta \in tr^\dagger(A_i) \ \&\ \alpha \,\mathbf{after}\, \beta \in tr^\dagger(P_i)$.

Revisiting the mutual exclusion example, we see that the two processes have fair closed expansions

$$\begin{aligned} P_0 &= cycle_0^\dagger; P_0 + \ldots + wait^\dagger; P_0 \\ P_1 &= cycle_1^\dagger; P_1 + \ldots + wait^\dagger; P_1 \end{aligned}$$

where

$$\begin{aligned} cycle_0^\dagger &= (grab; crit_0; rel; noncrit_0)^+ \\ cycle_1^\dagger &= (grab; crit_1; rel; noncrit_1)^+ \\ wait^\dagger &= idle^+ \end{aligned}$$

Taking the closure here allows us to elide the stuttering steps. We thus obtain a valid closed expansion for $P_0\|P_1$ with the following shape:

$$\begin{aligned} P_0\|P_1 = {} & (cycle_0^\dagger\|cycle_1^\dagger); P_0\|P_1 + \\ & (cycle_0^\dagger\|wait^\dagger); P_0\|P_1 + \\ & (wait^\dagger\|cycle_1^\dagger); P_0\|P_1 + \\ & (wait^\dagger\|wait^\dagger); P_0\|P_1 + \ \dots \end{aligned}$$

Forming the local expansion as before but taking closures again permits us to absorb the stutters, so that we end up with an expansion of shape

$$\begin{aligned} & \mathbf{local}\ sem = \mathbf{true\ in}\ (P_0\|P_1) \\ & \quad = \ (crit_0; noncrit_0)^+; \mathbf{local}\ sem = \mathbf{true\ in}\ (P_0\|P_1) \\ & \quad + \ \dots \end{aligned}$$

from which it follows that

$$\mathbf{local}\ s = \mathbf{true\ in}\ (P_0\|P_1) \supseteq (crit_0; noncrit_0)^\omega.$$

Closed trace semantics validates certain laws which can be used to justify common parallel programming idioms or “design patterns”. For instance the following law shows correctness of a form of *barrier synchronization*, in which processes periodically synchronize so as to stay “in phase” with each other:

$$\begin{aligned} & \mathbf{local}\ req, ack\ \mathbf{in}\ (P_1;\ req!\star;\ ack?\star;\ P_2)\|(Q_1;\ ack!\star;\ req?\star;\ Q_2) \\ & = \ (P_1\|Q_1);\ \mathbf{local}\ req, ack\ \mathbf{in}\ (P_2\|Q_2), \\ & \qquad\qquad \text{provided } req \text{ and } ack \text{ do not occur free in } P_1 \text{ or } Q_1. \end{aligned}$$

The value $\star$ is used here as a “token” solely to enforce synchronization. Note that the local channels are used in a simple manner: the system’s design ensures that the number of messages on each of these channels is bounded, so that when the $P_i$ and $Q_i$ are finite-state so is the entire system.

Barrier synchronization can also be used to enforce synchronization between recursive processes. If $P_1$ and $P_2$ are finite-state and do not use channels *req* and *ack*, and the process variables $p$, $q$ do not occur in $P_1$, $P_2$ respectively, the network

$$\mathbf{local}\ req, ack\ \mathbf{in}\ (\mathbf{rec}\ p.\, P_1;\ req!\star;\ ack?\star;\ p)\|(\mathbf{rec}\ q.\, P_2;\ ack!\star;\ req?\star;\ q)$$

can be proven equal to the recursive process $\mathbf{rec}\ r.\, (P_1\|P_2); r$ by means of the fair expansion law and the local expansion law.

## 9 Future Directions

If we allow the unconstrained use of integer- or channel-valued local variables it is easy to define processes which “count” or otherwise retain “memory” and therefore are not finite-state. For example, linking together two 1-place buffer processes (each of which is finite-state) then localizing their channel of communication produces an unbounded buffer, since the number of data items held in the local channel can grow without bound. To deal with such processes we can introduce *countable* families of mutually recursive process definitions.

We also intend to demonstrate the wider utility of these ideas and results by applying them to some examples from the literature. Fair expansion should be a powerful tool in analyzing networks built from finite-state processes, and we intend to investigate the possibility of implementing an automated model-checker based on our ideas.

## References

[1] Blamey, S., *The soundness and completeness of axioms for CSP processes*, in: Topology and category theory in computer science, Reed, M. and Roscoe, A.W., and Wachter, R. (editors), Oxford University Press, 1991.

[2] Brookes, S., *Full abstraction for a shared-variable parallel language*, LICS'93, IEEE Computer Society Press (1993), 98–109. Full version in: Information and Computation, vol 127, No. 2, Academic Press (June 1996).

[3] Brookes, S., *The essence of Parallel Algol*, LICS'96, IEEE Computer Society Press (1996) 164–173. Full version to appear: *Information and Computation*, 1998.

[4] Brookes, S., *Idealized CSP: Combining Procedures with Communicating Processes*, MFPS'97, Pittsburgh, March 1997. ENTCS 6, Elsevier Science. URL: `http://www.elsevier.nl/locate/entcs/volume6.html`.

[5] Brookes, S., *On the Kahn Principle and Fair Networks*, MFPS'98, Queen Mary Westfield College, May 1998. Submitted to *Theoretical Computer Science.*

[6] Formal Systems (Europe), Ltd., *Failures-Divergence Refinement: FDR2 Manual*, 1997.

[7] Hoare, C. A. R., *Communicating Sequential Processes*, Comm. ACM, 21(8):666–677 (1978).

[8] Hoare, C. A. R., **Communicating Sequential Processes**, Prentice-Hall (1985).

[9] Kahn, G., *The semantics of a simple language for parallel programming*, Information Processing '74, North Holland, 1974.

[10] Kahn, G. and MacQueen, D. B., *Coroutines and Networks of Parallel Processes*, Information Processing '77, North Holland, 1977.

[11] Milner, R., *A Complete Inference System for a Class of Regular Behaviors*, J. Comp. Syst. Sci. 28 (1984), pp. 439-466.

[12] Milner, R., **Communication and Concurrency**, Prentice-Hall (1989).

[13] Park, D., *On the semantics of fair parallelism.* In D. Bjørner, editor, **Abstract Software Specifications**, Springer-Verlag LNCS vol. 86 (1979), 504–526.

[14] Park, D., *Concurrency and Automata on Infinite Sequences*, LNCS vol. 104, Springer-Verlag (1980).

[15] Parrow, J., *Fairness Properties in Process Algebra*, Ph.D. thesis, Uppsala University (1985).

[16] Roscoe, A.W., **The Theory and Practice of Concurrency**, Prentice-Hall, 1998.

[17] Tarski, A., *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics, vol. 5, 1955.