Concurrent Transition Systems*

Eugene W. Stark
State University of New York at Stony Brook
Stony Brook, NY 11794

July 24, 1987

Abstract: *Concurrent transition systems* (CTS's), are ordinary nondeterministic transition systems that have been equipped with additional concurrency information, specified in terms of a binary *residual* operation on transitions. Each CTS C freely generates a *complete CTS* or *computation category* C^*, whose arrows are equivalence classes of finite computation sequences, modulo a congruence induced by the concurrency information. The categorical composition on C^* induces a "prefix" partial order on its arrows, and the *computations* of C are conveniently defined to be the ideals of this partial order. The definition of computations as ideals has some pleasant properties, one of which is that the notion of a *maximal ideal* in certain circumstances can serve as a replacement for the more troublesome notion of a *fair computation sequence.*

To illustrate the utility of CTS's, we use them to define and investigate a dataflow-like model of concurrent computation. The model consists of *machines*, which generalize the sequential machines of classical automata theory, and various *operations* (parallel product, input and output relabeling, and feedback) on machines that correspond to ways of combining machines into networks. Using our definition of computations as ideals, we define a natural notion of *observable equivalence* of machines, and show that it is the largest congruence, respecting parallel product and feedback, that does not relate two machines with distinct input/output relations. In an attempt to obtain information about the algebra of observable equivalence classes, we investigate a series of abstractions of the machine model, show that these abstractions respect the feedback operation, and characterize the homomorphic image of this operation in each case. A byproduct of our analysis is a simple characterization of a large class of processes with functional input/output behavior, and a proof that the feedback operation on such processes obeys Kahn's fixed-point principle.

1 Introduction

Labeled transition systems have been used as an operational semantics of concurrent processes. In typical formulations [9, 8], a labeled transition system is defined to be a tuple $M = (Q, q_0, \Sigma, \Delta)$, where Q is a set of *states*, q_0 is a distinguished *start state*, Σ is a set of *events*, not containing the distinguished symbol ϵ, and $\Delta \subseteq Q \times (\Sigma \cup \epsilon) \times Q$ is called the *transition relation.* Given such a transition system M, one can define a *computation sequence* of M to be a sequence of the form

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_n} q_n,$$

*This paper is a revised and expanded version of [35], which it supersedes. Some of the results presented here were reported, in an abbreviated form, in [36].

where each q_k is in Q, each σ_k is in $\Sigma \cup \{\epsilon\}$, and $(q_k, \sigma_{k+1}, q_{k+1}) \in \Delta$ for each k. The string $\sigma_1 \sigma_2 \ldots \sigma_n$ is called the *trace* of the computation. Here we regard the set Σ as embedded in the free monoid Σ^* in the obvious way, and regard ϵ as the identity element of this monoid; thus ϵ does not appear in a trace.

Formulations of labeled transition systems similar to the preceding have been used with some success in the study of concurrent programming languages such as CCS [27] and CSP [14], especially in the case where only finite computations are of interest. However, in the study of concurrency it is desirable to consider infinite computations as well, since processes in a concurrent system (e.g. an operating system) often are intended to run forever. Interesting properties of concurrent systems (such as guaranteed service of requests) cannot be properly expressed unless infinite computations are included in the underlying semantic model.

When one attempts to extend the use of transition systems to encompass the description of processes that run forever, things no longer work as smoothly as in the finite case. If we wish to define an operation of parallel composition, for example, which takes two transition systems and yields a new transition system that corresponds to the two original transition systems running in parallel, the linear nature of computation sequences forces us to use an "interleaved step" model of concurrency. Such an approach leads immediately to the so-called "fairness problem" [31]—a distinction must be drawn between "fair" computations, in which each process takes infinitely many steps, and "unfair" computations, in which one process performs only finitely many steps while the other enjoys infinitely many steps. Unfair computations can be screened out by applying some sort of fairness predicate to computations, or some sort of scheduling mechanism can be introduced to ensure that only fair computations are generated in the first place. Both approaches are mathematically inconvenient, since they involve the use of auxiliary notions (scheduling functions or predicates) not part of the basic transition system model, and these auxiliary notions tend to be ill-behaved (*e.g.* non-continuous).

For some time, the author has been interested in the possibility that by somehow viewing a transition system as being or generating a category, we might eliminate some of the difficulties associated with infinite computations. The basic idea would be to try to use the notion of "commuting paths" in a category to model the the various interleaved representations of a concurrent computation. It is clear that transition systems define categories in a natural way. Given a transition system $G = (Q, q_0, \Sigma, \Delta)$, we can define a "computation category" G^* whose object set is Q and which has as arrows from q to r all finite computation sequences of G that begin in state q and end in state r, with composition corresponding to concatenation of computation sequences. Infinite computation sequences with initial state q can be regarded as ideals (nonempty, downward-closed, directed subsets) of the set $G^*(q, -)$ of all arrows of G^* with domain q, partially ordered by prefix. Now, the categories G^* that result by this construction are free categories having no nontrivial commuting paths; thus we have apparently obtained little more than an insignificant restatement of the definition of computation sequence. However, from the new point of view it is interesting to ask whether some generalization of the usual notion of transition system might result in computation categories in which there are nontrivial commuting paths.

In this paper, we provide an affirmative answer to this question. We define the notion of a "concurrent transition system" (CTS), which consists of a (directed, multi-)graph, whose objects (nodes) are states and whose arrows (arcs) are transitions, which has been equipped with some additional concurrency information. Although not categories themselves, each CTS C freely generates a "complete CTS" or "computation category," C^* whose states (objects) are the same as those of C, and whose arrows (transitions) are equivalence classes of finite computation sequences of C. Each equivalence class can be thought of as the set of all interleaved views of a single underlying concurrent computation. The "prefix" relation induced by the categorical composition partially orders the set of arrows of the computation category. We define the "computations" of C, from start state q_0, to be the ideals of the set $C^*(q_0,\text{-})$ of all arrows of C^* with domain q_0, partially ordered by prefix. It follows from the ideal construction that the set of all computations from initial state q is an algebraic directed-complete partial order.

The main body of this paper is organized as follows: In Section 2, we define concurrent transition systems, give some examples, and derive some basic properties of CTS's and the category **CTS** in which they live. We establish the existence of the computation categories discussed above, and also of related structures called "computation diagrams," which generalize the familiar notion of "computation tree" for ordinary nondeterministic transition systems.

In Section 3, the theory developed in Section 2 is used to define a dataflow-like model of concurrent computation. The basic objects of the model are "machines," which are a CTS generalization of the sequential machines of classical automata theory. We define some operations (parallel product, input and output relabeling, and feedback) on machines that correspond to various ways of composing machines into networks. We define a natural notion of "observable equivalence" of machines, and show that it is the largest congruence on machines, respecting parallel product and feedback, that does not relate machines having distinct input/output relations. The "full abstraction problem" is defined as the problem of characterizing the structure of the quotient algebra of machines, modulo observable equivalence.

In Section 4, we perform a rather extensive analysis of the feedback operation, with the dual aims, of showing that our model is a reasonable one, and of attempting to make progress on the full abstraction problem. We define a sequence of abstraction mappings that starts with machines and ends with input/output relations. For each mapping, we prove a theorem that shows that the mapping is homomorphic with respect to the feedback operation. (The mapping to input/output relations is homomorphic only for the subclass of "Kahn" machines, which have continuous functions as their input/output behaviors.) Since some of the structures at the intermediate stages between machines and input/output relations are similar to models of concurrent processes that have been proposed in the literature, our analysis yields useful information about the relationships between these models.

In Section 5 we summarize what we have accomplished, discuss how our work is related to that of other authors, and mention possibilities for future research.

We assume that the reader is familiar with the basic notions of category theory. The necessary background can be found in [23, 2, 13]. We also assume some familiarity with the theory of algebraic directed-complete partial orders, as used in denotational semantics. Reference [11] provides background on this topic.

2 Concurrent Transition Systems

In this section, we define concurrent transition systems and derive some of their basic properties.

A *graph* is a tuple $G = (O, A, \mathrm{dom}, \mathrm{cod})$, where O is a set of *objects*, A is a set of *arrows*, and dom, cod are functions from A to O, which map each arrow to its *domain* and *codomain*, respectively. Arrows t, u of G are called *composable* if $\mathrm{cod}(t) = \mathrm{dom}(u)$ and *coinitial* if $\mathrm{dom}(t) = \mathrm{dom}(u)$. Let $\mathrm{Coin}(G)$ denote the set of all coinitial pairs of arrows of G.

If $G = (O, A, \mathrm{dom}, \mathrm{cod})$ is a graph, then define the *augmented graph* $G^\sharp = (O^\sharp, A^\sharp, \mathrm{dom}^\sharp, \mathrm{cod}^\sharp)$ to be the extension of G defined by: $O^\sharp = O \cup \{\Omega\}$, $A^\sharp = A \cup \{\omega_q : q \in O^\sharp\}$, $\mathrm{dom}^\sharp(\omega_q) = q$, and $\mathrm{cod}^\sharp(\omega_q) = \Omega$, where Ω is a new object not in O, and for each $q \in O^\sharp$, ω_q is a distinct new arrow not in A.

A *concurrent transition system* (CTS) is a structure $(G, \mathrm{id}, \uparrow)$, where

- $G = (O, A, \mathrm{dom}, \mathrm{cod})$ is a graph, called the *underlying graph*. The elements of $O^\sharp$ (resp. O) are called (resp. *proper*) *states* and the elements of $A^\sharp$ (resp. A) are called (resp. *proper*) *transitions*.
- $\mathrm{id} : O^\sharp \to A^\sharp$ maps each $q \in O^\sharp$ to a distinguished *identity transition* id_q.
- $\uparrow: \mathrm{Coin}(G^\sharp) \to A^\sharp$ is a function, called the *residual operation*. We write $t \uparrow u$ (read t "after" u) for $\uparrow(t, u)$.

These data are required to have the following properties:

1. For all $q \in O^\sharp$, and all coinitial $t, u \in A^\sharp$,
   (a) $\mathrm{dom}^\sharp(\mathrm{id}_q) = \mathrm{cod}^\sharp(\mathrm{id}_q) = q$;
   (b) $\mathrm{dom}^\sharp(t \uparrow u) = \mathrm{cod}^\sharp(u)$, and
   (c) $\mathrm{cod}^\sharp(t \uparrow u) = \mathrm{cod}^\sharp(u \uparrow t)$.
2. For all $t : q \to r$ in $A^\sharp$,
   (a) $\mathrm{id}_q \uparrow t = \mathrm{id}_r$;
   (b) $t \uparrow \mathrm{id}_q = t$; and
   (c) $t \uparrow t = \mathrm{id}_r$.
3. For all coinitial $t, u \in A^\sharp$, if $t \uparrow u$ and $u \uparrow t$ are both identities, then $t = u$.
4. For all coinitial $t, u, v \in A^\sharp$, $(v \uparrow t) \uparrow (u \uparrow t) = (v \uparrow u) \uparrow (t \uparrow u)$.

Axiom (4) can be visualized as shown in Figure 1.

In the sequel, we will drop the $\sharp$ from $\mathrm{dom}^\sharp$ and $\mathrm{cod}^\sharp$. Note that it automatically follows from the definition of a CTS that $\omega_q \uparrow t = \omega_r$ and $t \uparrow \omega_q = \omega_\Omega = \mathrm{id}_\Omega$ for all transitions $t : q \to r$, since for each q, the transition ω_q is the only transition with domain q and codomain Ω. Note also that states are actually not logically necessary in the definition of CTS, since they are in bijective correspondence with the set of identity transitions. We shall occasionally take advantage of this fact to give concise specifications of particular CTS's.

Coinitial transitions t, u of a CTS are called *consistent* if $t \uparrow u$ is a proper transition (equivalently, if $u \uparrow t$ is proper), otherwise they are called *conflicting*. A coinitial set of transitions is called *consistent* if it is pairwise consistent. In defining the operation $\uparrow$ for a CTS, we need only specify which coinitial pairs of proper transitions are consistent, and to give the definition of $\uparrow$ for such pairs, since the remaining cases are fixed by the CTS axioms.

A CTS is called *determinate* if every coinitial pair of proper transitions is consistent. A CTS is called *complete* if to every composable pair t, u of transitions there corresponds a transition v, called a *composite* of t and u, such that $t \uparrow v = \mathrm{id}_{\mathrm{cod}(v)}$ and $v \uparrow t = u$.

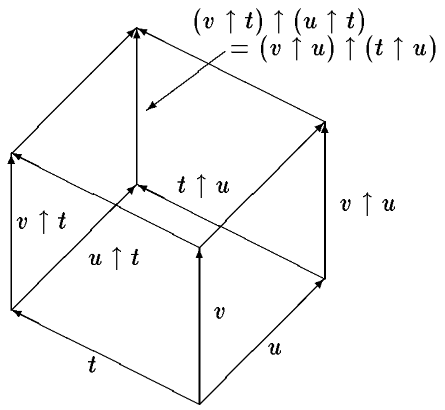

Figure 1: CTS Axiom (4)

Example 1 – CTS's From Graphs

Every graph G lifts to a CTS $\mathrm{Cts}(G)$ in a "minimally consistent" way. More precisely, suppose $G = (O, A, \mathrm{dom}, \mathrm{cod})$. Let $G' = (O, A', \mathrm{dom}', \mathrm{cod}')$ be defined so that $A' = A \uplus \{\mathrm{id}_q : q \in O\}$, and $\mathrm{dom}'(\mathrm{id}_q) = \mathrm{cod}'(\mathrm{id}_q) = q$ for all $q \in O$. Let $\uparrow$ be defined as follows:

$$t \uparrow u = \begin{cases} t, & \text{if } u = \mathrm{id}_{\mathrm{dom}(t)} \\ \mathrm{id}_{\mathrm{cod}(u)}, & \text{if } t = u \text{ or } t = \mathrm{id}_{\mathrm{dom}(t)} \\ \omega_{\mathrm{cod}(u)}, & \text{otherwise.} \end{cases}$$

It is easily verified that $C = (G', \mathrm{id}, \uparrow)$ is a CTS. Moreover, assuming an appropriate definition (provided in Section 2.2) of "CTS-morphism," it can be shown that the map Cts is the object map of a functor, left-adjoint to the forgetful functor taking each CTS to its underlying graph.

Example 2 – Trace Algebras

A *trace algebra* is a monoid X such that:

1. For all $t, u, v \in X$, if $tu = tv$, then $u = v$.

2. If $\preceq$ is the prefix relation induced by the monoid operation (*i.e.* $t \preceq u$ iff $\exists v(tv = u)$), then $\preceq$ is a partial order with respect to which each pair t, u with an upper bound has a least upper bound $t \vee u$.

The elements of a trace algebra are called *traces*.

One class of examples of trace algebras are those obtained from "concurrent alphabets" [26]. Formally, a *concurrent alphabet* is a pair $(\Sigma, ||)$, where Σ is a set and $||$ is an irreflexive, symmetric relation on Σ, called a *concurrency relation*. The concurrency relation induces a congruence $\sim$ on the free monoid Σ^*, such that two strings are congruent iff one can be transformed into the other by a finite sequence of steps in which pairs of adjacent concurrent symbols are permuted. The monoid $\Sigma^*/\sim$ is a trace algebra.

From a trace algebra X, we can construct a CTS C with one proper state, having the elements of X as its proper transitions, and in which the monoid identity ϵ is regarded as the single proper identity transition. If t, u is a pair of proper transitions of C (*i.e.* elements of X), then we define t and u to be consistent if they have a least upper bound $t \vee u$ as elements of X. For such a pair, we define $t \uparrow u$ to be the unique element of X with the property $u(t \uparrow u) = t \vee u$. It is straightforward to check that the CTS axioms are satisfied by these definitions. Moreover, the CTS C is complete, since the monoid operation yields, for each pair t, u of transitions, a transition tu with the properties $t \uparrow tu = \epsilon$ and $tu \uparrow t = u$.

Example 3 - CTS's From Petri Nets

In [38] a "net" is defined to be a bipartite directed graph $N = (B, E; F)$, where the set of nodes $B \cup E$ is partitioned into a set E of *events* and a set B of *conditions*, and the relation $F \subseteq (B \times E) \cup (E \times B)$, is called the *flow relation.* A *case* of N is a set of conditions. For each event $e \in E$, the set $\mathrm{pre}(e)$ of *preconditions* of e is the set of all $b \in B$ such that $(b, e) \in F$, and the set *post*(e) of *postconditions* of e is the set of all $b \in B$ such that $(e, b) \in F$. A set u of events is called *independent* if for each pair e_1, e_2 of elements of u, the sets $\mathrm{pre}(e_1) \cup \mathrm{post}(e_1)$ and $\mathrm{pre}(e_2) \cup \mathrm{post}(e_2)$ are disjoint.

The *transition relation* of N is the set of all triples (c, u, c'), where c and c' are cases of N, and u is an independent set of events of N such that:

1. For all $e \in u$, $\mathrm{pre}(e) \subseteq c$.

2. For all $e \in u$, $\mathrm{post}(e) \cap c = \emptyset$.

3. The case c' is obtained from c by removing all preconditions of events in u, and then adding all postconditions of events in u.

We now obtain a CTS from N as follows: Take as states the cases of N. Take as proper transitions the elements of the transition relation of N, defining $\mathrm{dom}(c, u, c') = c$ and $\mathrm{cod}(c, u, c') = c'$, and regarding the transitions $(c, \emptyset, c)$ as identities. Define coinitial pairs (c, u, d) and (c, v, d') of proper transitions to be consistent iff $u \cup v$ is independent. For such pairs, define

$$(c, u, d) \uparrow (c, v, d') = (d', u \backslash v, e),$$

where e is obtained from d' by removing all preconditions of events in $u \backslash v$ and then adding all postconditions of events in $u \backslash v$. That these definitions satisfy the CTS axioms is easily checked.

Example 4 - CTS's From λ-Calculus

Let Λ be the set of terms of the pure λ-calculus [4], and let $\rightarrow$ denote reduction with respect to rule (β). If C is a set of redexes in a term q, then a *derivation relative to* C is a derivation $q = q_0 \rightarrow q_1 \rightarrow \ldots \rightarrow q_n$, in which the redex contracted at each step is either in C or is a residual (in Church's original sense) of a redex in C. A derivation relative to C is *complete* if the set of residuals of C in q_n is empty. Given a set C of redexes in a term q, it can be shown that there is a fixed upper bound on the length of a derivation from q relative to C, and that all complete derivations from q relative to C result in the same term. It therefore makes sense to write $q \xrightarrow{C} r$, if C is a set of redexes in q and any complete derivation relative to C results in r. Let us call such triples $q \xrightarrow{C} r$ *transitions.* Suppose t is a transition $q \xrightarrow{C} r$ and u is a transition $q \xrightarrow{D} s$. It can be shown that the same set C/d of residuals of redexes in C is obtained for all complete derivations

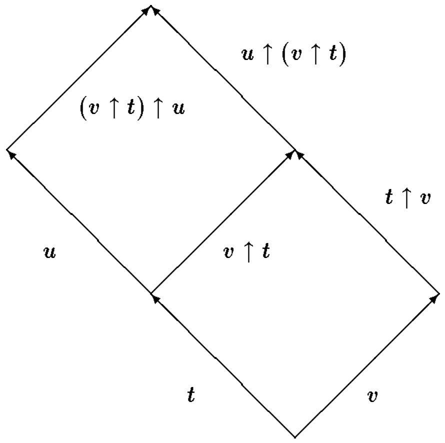

Figure 2:

d relative to D. Let C/D denote this set. It therefore makes sense to define the *residual* t/u of transition t with respect to u to be the transition $s \xrightarrow{C/D} p$, where p is the unique result of a complete derivation from s relative to C/D.

We now define a CTS whose state set is Λ, and whose transitions are all transitions t as defined above. The domain $\mathrm{dom}(t)$ of a transition $t : q \xrightarrow{C} r$ is the term q and the codomain $\mathrm{cod}(t)$ of t is the term r. The identity transitions are those of the form $q \xrightarrow{\emptyset} q$, all coinitial pairs of proper transitions are consistent, and we define and we define $t \uparrow u = t/u$.

That the above definitions satisfy the CTS axioms follows from results of Lèvy [22, 5]. A similar construction can be used to obtain CTS's from left-linear term-rewriting systems without critical pairs [15].

2.1 Consequences of the Axioms

Define a relation $\preceq$ on the transitions of a CTS by: $t \preceq u$ iff t, u are coinitial and $t \uparrow u = \mathrm{id}_{\mathrm{cod}(u)}$. We call $\preceq$ the *prefix relation.*

Lemma 2.1.1 *The prefix relation is a partial order.*

Proof – Reflexivity holds because $t \uparrow t = \mathrm{id}_{\mathrm{cod}(t)}$ by axiom (2c).

To show transitivity, suppose $t \preceq u$ and $u \preceq v$. Then $t \uparrow u$ is an identity, so $(t \uparrow u) \uparrow (v \uparrow u)$ is an identity by axiom (2a). Since $(t \uparrow u) \uparrow (v \uparrow u) = (t \uparrow v) \uparrow (u \uparrow v)$ by axiom (4), it follows that $(t \uparrow v) \uparrow (u \uparrow v)$ is an identity. But $(u \uparrow v)$ is an identity because $u \preceq v$, hence $t \uparrow v$ is an identity by axiom (2b).

Finally, $\preceq$ is antisymmetric because if $t \preceq u$ and $u \preceq t$, then $t \uparrow u$ and $u \uparrow t$ are identities, hence $t = u$ by axiom (3). ∎

Lemma 2.1.2 *If a composite of t and u exists, then it is unique.*

Proof – Suppose v and v' are composites of t and u. Then $t \uparrow v'$ is an identity, so $v \uparrow v' = (v \uparrow v') \uparrow (t \uparrow v')$ by axiom (2b). By axiom (4), this is equal to $(v \uparrow t) \uparrow (v' \uparrow t) = u \uparrow u$, which is an identity by axiom (2c). A symmetric argument shows that $v' \uparrow v$ is an identity, so $v = v'$ by axiom (3). ∎

We use tu to denote the composite of t and u, when it exists.

Lemma 2.1.3 *Suppose t and v are coinitial, and the composite tu of t and u exists. Then*

1. $v \uparrow tu = (v \uparrow t) \uparrow u$.
2. $tu \uparrow v = (t \uparrow v)(u \uparrow (v \uparrow t))$.

Proof – (See Figure 2.)

(1) $v \uparrow tu = (v \uparrow tu) \uparrow (t \uparrow tu) = (v \uparrow t) \uparrow (tu \uparrow t) = (v \uparrow t) \uparrow u$.

(2) Since $(t \uparrow v) \uparrow (tu \uparrow v) = (t \uparrow tu) \uparrow (v \uparrow tu)$, which is an identity, and $(tu \uparrow v) \uparrow (t \uparrow v) = (tu \uparrow t) \uparrow (v \uparrow t) = u \uparrow (v \uparrow t)$, the result follows. ∎

Lemma 2.1.4 *Composition obeys the following laws:*

1. *For all t, $t = \mathrm{id}_{\mathrm{dom}(t)} t = t\, \mathrm{id}_{\mathrm{cod}(t)}$.*
2. (a) *If tu and $(tu)v$ exist, then uv and $t(uv)$ exist, and $(tu)v = t(uv)$.*
   (b) *If tu, uv, and $t(uv)$ exist, then $(tu)v$ exists and $(tu)v = t(uv)$.*
3. *If tu and tv exist, and $tu = tv$, then $u = v$.*

Proof – (1) follows directly from the definition of composite.

To show (2a), suppose tu and $(tu)v$ exist. Then by Lemma 2.1.3, $(tu)v \uparrow t = (tu \uparrow t)(v \uparrow (t \uparrow tu))$. But $tu \uparrow t = u$ and $t \uparrow tu$ is an identity, so $(tu)v \uparrow t = uv$. Since by Lemma 2.1.3, $t \uparrow (tu)v = (t \uparrow tu) \uparrow v$, which is an identity, it follows that $(tu)v = t(uv)$.

To show (2b), suppose tu, uv and $t(uv)$ exist. By Lemma 2.1.3,

$$t(uv) \uparrow tu = (t \uparrow tu)(uv \uparrow (tu \uparrow t)),$$

which is just v. Also,

$$tu \uparrow t(uv) = (t \uparrow t(uv))(u \uparrow (t(uv) \uparrow t)),$$

which is an identity, so $t(uv) = (tu)v$.

For (3), suppose $tu = tv$, so that $tu \uparrow tv$ and $tv \uparrow tu$ are identities. Then $u \uparrow v = (tu \uparrow t) \uparrow (tv \uparrow t) = (tu \uparrow tv) \uparrow (t \uparrow tv)$, which is an identity. Similarly, $v \uparrow u$ is an identity, so $u = v$. ∎

We say that a transition v is a *join* of the coinitial transitions t, u if $t \preceq v$, $u \preceq v$, $v \uparrow t = u \uparrow t$, and $v \uparrow u = t \uparrow u$.

Lemma 2.1.5 *A transition v is a join of t and u iff $v = t(u \uparrow t)$.*

Proof – If v is a join of t and u, then $t \uparrow v$ is an identity and $v \uparrow t = u \uparrow t$, so $v = t(u \uparrow t)$. Conversely, if $v = t(u \uparrow t)$, then $t \preceq v$, $v \uparrow t = u \uparrow t$, $u \uparrow v = (u \uparrow t) \uparrow (u \uparrow t)$, which is an identity, so $u \preceq v$, and $v \uparrow u = (t \uparrow u)((u \uparrow t) \uparrow (u \uparrow t)) = (t \uparrow u)$. Hence v is a join of t and u. ∎

It follows from the preceding lemma and the uniqueness of composites that a join of t and u, when it exists, is unique, and we denote it by $t \vee u$. Moreover, if $t \vee u$ exists, then we have the equality $t(u \uparrow t) = t \vee u = u(t \uparrow u)$.

Lemma 2.1.6 *Suppose $t \vee u$ exists. Then $t \vee u$ is the least upper bound of t and u under $\preceq$.*

Proof – By definition, $t \vee u$ is an upper bound of t and u under $\preceq$. Suppose l is any upper bound of t and u. Let $v = l \uparrow t$ and $w = l \uparrow u$, so that $tv = l = uw$. Now, $(u \uparrow t) \uparrow v = (u \uparrow t) \uparrow (tv \uparrow t) = (u \uparrow tv) \uparrow (t \uparrow tv) = (u \uparrow uw) \uparrow (t \uparrow tv)$, which is an identity, so $u \uparrow t \preceq v$. Let $m = v \uparrow (u \uparrow t)$, so that $v = (u \uparrow t)m$. It then follows that $l = tv = t(u \uparrow t)m = (t \vee u)m$, so that $(t \vee u) \preceq l$. ∎

2.2 The Category CTS

If $G = (O, A, \mathrm{dom}, \mathrm{cod})$ and $G' = (O', A', \mathrm{dom}', \mathrm{cod}')$ are graphs, then a *graph morphism* from G to G' is a pair of maps $\rho = (\rho_\mathrm{o}, \rho_\mathrm{a})$, where $\rho_\mathrm{o} : O \to O'$ and $\rho_\mathrm{a} : A \to A'$, such that $\mathrm{dom}' \circ \rho_\mathrm{a} = \rho_\mathrm{o} \circ \mathrm{dom}$ and $\mathrm{cod}' \circ \rho_\mathrm{a} = \rho_\mathrm{o} \circ \mathrm{cod}$. In the sequel, we will drop the notational distinction between ρ_o and ρ_a, writing simply ρ in both cases.

A *CTS-morphism* from a CTS $C = (G, \mathrm{id}, \uparrow)$ to a CTS $C' = (G', \mathrm{id}', \uparrow')$ is a graph morphism $\rho : G \to G'$, with the following properties:

1. For all proper states q of C, $\rho(\mathrm{id}_q) = \mathrm{id}'_{\rho(q)}$.

2. If t, u are consistent proper transitions of C, then $\rho(t \uparrow u) = \rho(t) \uparrow' \rho(u)$.

It will be useful to think of a morphism $\rho : C \to C'$ as extended to all states and transitions of C (not just the proper ones), according to the definitions $\rho(\Omega) = \Omega$, and $\rho(\omega_q) = \omega_{\rho(q)}$. The set of all CTS's forms a category **CTS**, when equipped with the CTS-morphisms as arrows.

In the sequel, the term "morphism" will mean "CTS-morphism," unless otherwise specified.

Lemma 2.2.1 *Suppose $\rho : C \to C'$ is a morphism. Then*

1. $\rho(tu) = \rho(t)\rho(u)$ whenever tu exists and is a proper transition.

2. $\rho(t \vee u) = \rho(t) \vee \rho(u)$ whenever $t \vee u$ exists and is a proper transition.

Proof – (1) Since t and tu are consistent, $\rho(t) \uparrow \rho(tu) = \rho(t \uparrow tu)$, which is an identity. Also, $\rho(tu) \uparrow \rho(t) = \rho(tu \uparrow t) = \rho(u)$, so $\rho(tu) = \rho(t)\rho(u)$.

(2) If $t \vee u$ exists and is a proper transition, then t and u are consistent, hence t and $t \vee u$ are consistent. Thus, $\rho(t \vee u) \uparrow \rho(t) = \rho((t \vee u) \uparrow t) = \rho(u \uparrow t) = \rho(u) \uparrow \rho(t)$. Also, $\rho(t) \uparrow \rho(t \vee u) = \rho(t \uparrow (t \vee u))$, which is an identity. Symmetric reasoning shows that $\rho(t \vee u) \uparrow \rho(u) = \rho(t) \uparrow \rho(u)$ and $\rho(u) \uparrow \rho(t \vee u)$ is an identity. It follows that $\rho(t \vee u) = \rho(t) \vee \rho(u)$. ∎

Lemma 2.2.2 *The forgetful functor from* **CTS** *to* **Graph** *that takes each CTS C to its underlying graph has a left adjoint, whose object map takes each graph G to the corresponding "minimally consistent" CTS* $\mathrm{Cts}(G)$.

Proof – Straightforward. ∎

Note that since the forgetful functor from **CTS** to **Graph** is a right adjoint, and hence preserves limits, from the previous lemma we know that any limits existing in **CTS** must be lifted from corresponding limits in **Graph**. In fact, we have:

Theorem 2.1 *The following constructions in* **Graph** *lift to* **CTS***:*

1. *Small limits.*

2. *Small coproducts.*

Proof – (1) It suffices to show that small products and equalizers of pairs of morphisms lift from **Graph** to **CTS**, since a standard category-theoretic result (see, *e.g.* [23], p. 109) states that the existence of small products and equalizers of pairs of morphisms implies the existence of all small limits. The obvious constructions work in each case.

(2) The obvious construction works. ∎

We do not know whether coequalizers of all pairs of morphisms exist in **CTS**. However, we can prove a somewhat weaker result, and the quotient construction it requires will be of use in Section 2.3.

Suppose C is a CTS. A *strong congruence* on C is an equivalence relation $\sim$ on the set of proper transitions of C, such that

1. For all transitions t, t', u, u' of C, if $t \sim t'$, $u \sim u'$, and t, u are consistent, then t', u' are consistent, and $t \uparrow u \sim t' \uparrow u'$.

2. For all coinitial transitions t, u of C, if $t \uparrow u$ and $u \uparrow t$ are both $\sim$-related to identities, then $t \sim u$.

Since the defining properties of a strong congruence are evidently closure properties, it follows that every binary relation on the transitions of a CTS C is contained in a least strong congruence on C.

If $\sim$ is a strong congruence on C, then we may form the *quotient CTS* $C/\sim$ as follows:

- Take the proper states of C as the proper states of $C/\sim$.

- Define the proper transitions of $C/\sim$ to be the $\sim$-equivalence classes. We write $[t]$ for the equivalence class of t, and we define the identities of $C/\sim$ to be the equivalence classes of identities of C. Define $\mathrm{dom}([t]) = \mathrm{dom}(t)$ and $\mathrm{cod}([t]) = \mathrm{cod}(t)$.

- Define $[t], [u]$ to be consistent iff t, u are consistent, in which case we define $[t] \uparrow [u] = [t \uparrow u]$.

It is easily shown, using the properties of a strong congruence, these definitions are independent of the choice of t and u. It is also straightforward to check that $C/\sim$ satisfies the CTS axioms.

A *strong morphism* is a morphism $\rho : C \to D$ such that for all transitions t, t', u of C, if t, u are consistent and $\rho(t) = \rho(t')$, then t', u are consistent. The *quotient map* from C to $C/\sim$ is the function η that takes each transition t of C to its $\sim$-equivalence class $[t]$. It is clear that η is a strong morphism.

Theorem 2.2 *Suppose C is a CTS, and R is a binary relation on the transitions of C. Let $\sim$ be the least strong congruence on C that contains R, and let $\eta : C \to C/\sim$ be the quotient map. Then for each strong morphism $\rho : C \to D$, with the property that tRu implies $\rho(t) = \rho(u)$ for all transitions t, u of C, there exists a unique morphism $\sigma : (C/\sim) \to D$, such that $\rho = \sigma \circ \eta$.*

Proof – Straightforward. ▮

A *CTS with start state* is a pair (C, ι), where C is a CTS, and $\iota : \mathbf{1} \to C$ is a morphism, called the *start state map*, from the terminal object $\mathbf{1}$ of CTS to C. The *start state* of (C, ι) is the image under ι of the single proper state of $\mathbf{1}$. If (C, ι) and (C', ι') are CTS's with start state, then a *morphism* from (C, ι) to (C', ι') is a morphism $\rho : C \to C'$ such that $\rho \circ \iota = \iota'$. Let $\mathbf{CTS}_\iota$ denote the category of CTS's with start state, and their morphisms. It is a straightforward consequence of the results for **CTS** that the category $\mathbf{CTS}_\iota$ has all small limits. It can also be shown that $\mathbf{CTS}_\iota$ has all small coproducts.

2.3 Computation Categories

Define a *computation category* to be a small category C with the following properties:

1. C has a terminal object.
2. Every arrow of C is an epimorphism.
3. Every isomorphism of C is an identity.
4. C has a pushout for every coinitial pair of arrows.

Theorem 2.3 *Suppose $C = (G, \mathrm{id}, \uparrow)$ is a complete CTS, and let $\cdot$ denote the composition operation of C. Then $C' = (G^\sharp, \mathrm{id}, \cdot)$ is a computation category. Conversely, suppose $C' = (G', \mathrm{id}, \cdot)$ is a computation category. Since the terminal object of C' is unique by property (3), we may regard G' as an augmented graph $G^\sharp$. For coinitial arrows t, u, let $t \uparrow u$ denote the arrow opposite t in the pushout square determined by t and u. Then $C = (G, \mathrm{id}, \uparrow)$ is a complete CTS.*

Proof – ($\Leftarrow$) If a small category C' is given with properties (1)-(4), then standard category-theoretic arguments suffice to show that $(G, \mathrm{id}, \uparrow)$ satisfies the axioms for a complete CTS.

($\Rightarrow$) Conversely, given a complete CTS C, it follows from completeness and Lemma 2.1.4 that C' is a category in which every arrow is an epimorphism. The state Ω is clearly a terminal object of C'. If v is an isomorphism of C', with inverse v', then $v \preceq vv' = \mathrm{id}_{\mathrm{dom}(v)}$ and $\mathrm{id}_{\mathrm{dom}(v)} \preceq v$, so $v = \mathrm{id}_{\mathrm{dom}(v)} = v'$ by Lemma 2.1.1. The completeness of C implies that every coinitial pair of arrows t, u has a join $t \vee u$, which is a least upper bound of t and u by Lemma 2.1.6. Thus, every upper bound of t and u factors through $t \vee u$. The uniqueness of such factorizations follows from the fact that every arrow is an epimorphism. Since $t(u \uparrow t) = t \vee u = u(t \uparrow u)$, it is now immediate that $t, u, (u \uparrow t)$, and $(t \uparrow u)$ form a pushout square in C'. ▮

We now show that every CTS C freely generates a complete CTS C^*, which has the same states as C and whose transitions are equivalence classes of finite composable sequences of transitions of C. The construction generalizes the construction of the free category G^* generated by a graph G. The construction was discovered by Lévy [22] in the setting of the λ-calculus, and adapted in [5, 15] to the cases of recursive programs and left-linear term-rewriting systems without critical

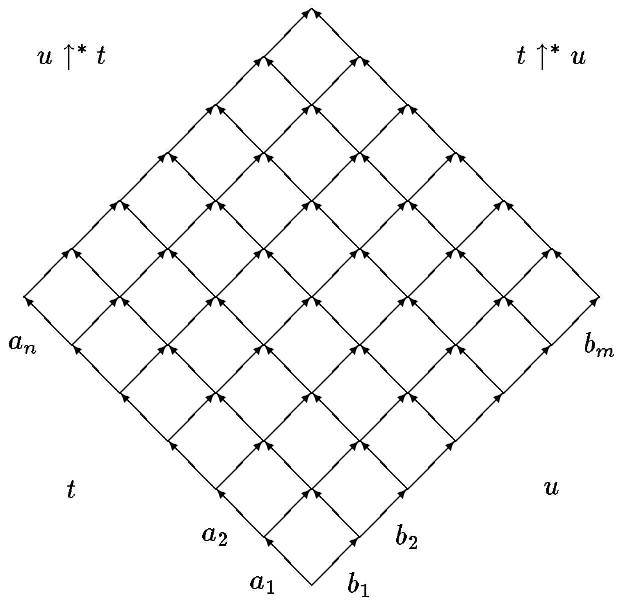

Figure 3: Extension of $\uparrow$ to $\uparrow^*$

pairs. Consideration of problems in the theory of concurrency led to its independent rediscovery by the author in the present axiomatic setting.

Suppose $C = (G, \mathrm{id}, \uparrow)$ is a CTS. Let G^* denote the free category (objects=objects of G, arrows=finite composable sequences of arrows of G) generated by G. If $t = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$ is an arrow of G^*, then let $|t|$ denote the length n of t. For $k > 0$, let id_q^k denote the arrow of G^* which is the sequence consisting of k copies of id_q. By convention, we define id_q^0 to be the identity for state q in G^*.

Let $\uparrow^*$ be the extension to $(G^*)^\sharp$ of the operation $\uparrow$ on $G^\sharp$, defined recursively by the following properties (see Figure 3):

1. $\mathrm{id}_q^0 \uparrow^* t = \mathrm{id}_r^0$ and $t \uparrow^* \mathrm{id}_q^0 = t$ for all $t : q \to r$ in G^*.

2. For all t in G^* and all $a, b \in G$, with a, b coinitial and a, t composable,

$$at \uparrow^* b = \begin{cases} (a \uparrow b)(t \uparrow^* (b \uparrow a)), & \text{if } a, b \text{ and } t, b \uparrow a \text{ are consistent} \\ \omega_{\mathrm{cod}(b)}, & \text{otherwise.} \end{cases}$$

3. For all t, u in G^* with $|u| > 0$, and all $b \in G$, with b, t coinitial and b, u composable,

$$t \uparrow^* bu = (t \uparrow^* b) \uparrow^* u$$

4. $\omega_q \uparrow^* t = \omega_r$ and $t \uparrow^* \omega_q = \omega_\Omega$ for all $t : q \to r$ in $(G^*)^\sharp$.

Lemma 2.3.1

1. *For all objects q of $(G^*)^\sharp$, all coinitial arrows t, u of $(G^*)^\sharp$, and all $k \geq 0$,*

   (a) $\mathrm{dom}(\mathrm{id}_q^k) = \mathrm{cod}(\mathrm{id}_q^k) = q$;

(b) $\mathrm{dom}(t \uparrow^* u) = \mathrm{cod}(u)$; *and*

(c) $\mathrm{cod}(t \uparrow^* u) = \mathrm{cod}(u \uparrow^* t)$.

2. *For all arrows* $t : q \to r$ *of* $(G^*)^\sharp$, *and all* $k \geq 0$,

   (a) $\mathrm{id}_q^k \uparrow^* t = \mathrm{id}_r^k$;

   (b) $t \uparrow^* \mathrm{id}_q^k = t$;

   (c) $t \uparrow^* t = \mathrm{id}_r^{|t|}$;

3. *For all coinitial arrows* t, u, v *of* $(G^*)^\sharp$, $(v \uparrow^* t) \uparrow^* (u \uparrow^* t) = (v \uparrow^* u) \uparrow^* (t \uparrow^* u)$.

Proof – Straightforward induction arguments using the properties of $\uparrow$ and the definition of $\uparrow^*$. ∎

Let $\mathrm{Cts}(G^*)$ denote the "minimally consistent" CTS with G^* as its underlying graph, as defined in Section 2, Example 1.

Let $\sim$ be the binary relation on the arrows of G^* defined by: $t \sim u$ iff $t \uparrow^* u = \mathrm{id}_r^{|t|}$ and $u \uparrow^* t = \mathrm{id}_r^{|u|}$.

Lemma 2.3.2 *The relation* $\sim$ *is a strong congruence on* $\mathrm{Cts}(G^*)$.

Proof – It is clear that $\sim$ is symmetric. It is reflexive because $t \uparrow^* t = \mathrm{id}_r^{|t|}$ by Lemma 2.3.1(2c). Transitivity follows from Lemma 2.3.1(3), as in the proof of Lemma 2.1.1.

(1) Suppose transitions t, t', u, u' of $\mathrm{Cts}(G^*)$ are such that $t \sim t'$, $u \sim u'$, and t, u are consistent. We show that t', u are consistent and $t \uparrow^* u \sim t' \uparrow^* u$; a similar argument (which we omit) then shows that t', u' are consistent and $t' \uparrow^* u \sim t' \uparrow^* u'$. By Lemma 2.3.1(3), $(t' \uparrow^* u) \uparrow^* (t \uparrow^* u) = (t' \uparrow^* t) \uparrow^* (u \uparrow^* t)$, which equals $\mathrm{id}_t^{|t'|}$ by Lemma 2.3.1(2a) and the assumption that $t \sim t'$. Similarly, $(t \uparrow^* u) \uparrow^* (t' \uparrow^* u) = (t \uparrow^* t') \uparrow^* (u \uparrow^* t') = \mathrm{id}_t^{|t|}$, hence $t \uparrow^* u \sim t' \uparrow^* u$. Since t and u are consistent by assumption, it follows that t' and u are consistent.

(2) First note that it is immediate from Lemma 2.3.1(2a,b) that $t \sim \mathrm{id}_q^0$ holds for a transition of $\mathrm{Cts}(G^*)$ iff $t = \mathrm{id}_q^{|t|}$. Now, suppose $t \uparrow u$ and $u \uparrow t$ are both $\sim$-related to identities. Then we must have $t \uparrow u = \mathrm{id}_{\mathrm{cod}(u)}^{|t|}$ and $u \uparrow t = \mathrm{id}_{\mathrm{cod}(t)}^{|u|}$. But this states exactly that $t \sim u$. ∎

Define the *completion* C^* of C to be the quotient $\mathrm{Cts}(G^*)/\sim$. That C^* is a complete CTS is immediate from the completeness of $\mathrm{Cts}(G^*)$ and Lemma 2.2.1.

Theorem 2.4 *The map taking a CTS C to its completion C^* is the object map of a functor from* **CTS** *to the full subcategory* **CCTS** *of* **CTS** *whose objects are the complete CTS's, and this functor is left-adjoint to the inclusion of* **CCTS** *in* **CTS**.

Proof – Let $\mu_C : C \to C^*$ take each transition t of C to its equivalence class $[t]$; then μ_C is obviously a morphism. To prove the theorem, it suffices to show that to each morphism ρ from C to a complete CTS D, there is a unique morphism $\rho^* : C^* \to D$ satisfying $\rho^* \circ \mu_C = \rho$.

We first note that a straightforward induction using Lemma 2.3.1 establishes the following fact: If

$$a_1 a_2 \ldots a_n \uparrow^* b_1 b_2 \ldots b_m = c_1 c_2 \ldots c_n,$$

then

$$\rho(a_1)\rho(a_2)\dots\rho(a_n) \uparrow \rho(b_1)\rho(b_2)\dots\rho(b_m) = \rho(c_1)\rho(c_2)\dots\rho(c_n).$$

Now, every proper transition of C^* is either an identity or of the form $[a_1a_2\dots a_n]$, where $n > 0$ and each a_i is a proper transition of C. By Lemma 2.2.1, any morphism $\rho^* : C^* \to D$ must satisfy $\rho^*([a_1a_2\dots a_n]) = \rho^*([a_1])\rho^*([a_2])\dots\rho^*([a_n])$. The condition $\rho^* \circ \mu_C = \rho$ implies in addition that $\rho^*([a_i]) = \rho(a_i)$, hence $\rho^*([a_1a_2\dots a_n]) = \rho(a_1)\rho(a_2)\dots\rho(a_n)$. Thus, there can be at most one morphism $\rho^* : C^* \to D$ satisfying $\rho^* \circ \mu_C = \rho$.

To show that ρ^* exists, it suffices to show that for all proper transitions $a_1, a_2, \dots, a_n$ and $b_1, b_2, \dots, b_m$ of C, if $[a_1a_2\dots a_n] = [b_1b_2\dots b_m]$, then $\rho(a_1)\rho(a_2)\dots\rho(a_n) = \rho(b_1)\rho(b_2)\dots\rho(b_m)$. But if $[a_1a_2\dots a_n] = [b_1b_2\dots b_m]$, then $a_1a_2\dots a_n \uparrow^* b_1b_2\dots b_m = \mathrm{id}^n_{\mathrm{cod}(b_m)}$ and $b_1b_2\dots b_m \uparrow^* a_1a_2\dots a_n = \mathrm{id}^m_{\mathrm{cod}(a_n)}$. Applying the fact noted above, we see that

$$\rho(a_1)\rho(a_2)\dots\rho(a_n) \uparrow \rho(b_1)\rho(b_2)\dots\rho(b_m) \text{ and } \rho(b_1)\rho(b_2)\dots\rho(b_m) \uparrow \rho(a_1)\rho(a_2)\dots\rho(a_n)$$

are identities, showing $\rho(a_1)\rho(a_2)\dots\rho(a_n) = \rho(b_1)\rho(b_2)\dots\rho(b_m)$.

Finally, to show that the function ρ^* is a morphism, we must show that it preserves identities and translation. Each identity of C^* is of the form $[\mathrm{id}^n_q]$, where id_q is an identity of C, and we must have $\rho^*([\mathrm{id}^n_q]) = \rho(\mathrm{id}_q)^n = \mathrm{id}_{\rho(q)}$. Moreover, if $[a_1a_2\dots a_n]$ and $[b_1b_2\dots b_m]$ are consistent, then using the fact noted above shows that

$$\begin{aligned}
\rho^*([a_1a_2\dots a_n] \uparrow [b_1b_2\dots b_m]) &= \rho^*([a_1a_2\dots a_n \uparrow^* b_1b_2\dots b_m]) \\
&= \rho(a_1)\rho(a_2)\dots\rho(a_n) \uparrow \rho(b_1)\rho(b_2)\dots\rho(b_m) \\
&= \rho^*([a_1a_2\dots a_n]) \uparrow \rho^*([b_1b_2\dots b_m]),
\end{aligned}$$

as required. ∎

2.4 Computation Diagrams

In this section we generalize to CTS's the notions of "computation tree" and "computation" for ordinary transition systems.

A *computation diagram* is a CTS D, whose completion D^* is a poset category having an initial (least) object. We regard the class of all computation diagrams as a full subcategory **Diag** of $\mathbf{CTS}_\iota$, by identifying a computation D with the corresponding CTS with start state (D, ι), whose start state is the initial object of D.

Define the *elements* of a computation diagram D to be the proper initial arrows of D^*. Note that the elements of D are in bijective correspondence with the states of D. Define the *successor relation* $\prec_1$ on the elements of D by $t \prec_1 u$ iff $u \uparrow t$ is a transition of D. The successor relation is easily seen to be an irreflexive relation, whose reflexive, transitive closure is the prefix relation $\preceq$ on the set of elements of D.

Given a CTS with start state (C, ι), let C^* be the completion of C. Define the CTS $\mathrm{Diag}(C, \iota)$ to have as proper states all transitions t of C^* whose domain is the start state of (C, ι), and as transitions pairs (t, u) of states of $\mathrm{Diag}(C, \iota)$, such that $t \preceq u$ and $u \uparrow t$ is a transition of C. Define $\mathrm{dom}(t, u) = t$ and $\mathrm{cod}(t, u) = u$. Let (t, u) and (t, v) be consistent in $\mathrm{Diag}(C, \iota)$ iff u and v are consistent in C, in which case define $(t, u) \uparrow (t, v) = (v, u \vee v)$. It is easily verified that $\mathrm{Diag}(C, \iota)$ is a computation diagram, and we call it the *computation diagram of* (C, ι).

Theorem 2.5 *The map, taking a CTS with start state (C, ι) to its computation diagram* $\mathrm{Diag}(C, \iota)$*, is the object map of a functor* $\mathrm{Diag} : \mathbf{CTS}_\iota \to \mathbf{Diag}$*, which is right-adjoint to the inclusion of* $\mathbf{Diag}$ *in* $\mathbf{CTS}_\iota$*.*

Proof – Let $\pi_C : \mathrm{Diag}(C, \iota) \to (C, \iota)$ take each transition (t, u) of $\mathrm{Diag}(C, \iota)$ to the corresponding transition $u \uparrow t$ of C. It is straightforward to check that π_C is a $\mathbf{CTS}_\iota$-morphism, and that every morphism ρ from a computation diagram D to (C, ι) factors uniquely through π_C. ▮

Recall that an *ideal* of a partially ordered set $(K, \preceq)$ is a subset J of K that is:

1. (Nonempty): $J \neq \emptyset$.
2. (Downward-closed): If $t \in J$ and $u \in K$ are such that $u \preceq t$, then $u \in J$.
3. (Directed): If $t \in J$ and $u \in J$, then there exists $v \in J$ such that $t \preceq v$ and $u \preceq v$.

An ideal J of K is *principal* if it is the set of all elements below some element t of K.

We define a *computation* of a CTS with start state (C, ι) to be an ideal of the set of elements of its computation diagram $\mathrm{Diag}(C, \iota)$, under the ordering $\preceq$. A computation of C is called *finite* if it is a principal ideal, otherwise it is called *infinite*. Note that the computations of (C, ι) are in natural bijective correspondence with the determinate subdiagrams of $\mathrm{Diag}(C, \iota)$. Although we could have defined the computations of (C, ι) to be the determinate subdiagrams of $\mathrm{Diag}(C, \iota)$, it will be convenient later on to think of a computation of (C, ι) as a set of transitions of C^*, rather than as a subdiagram of $\mathrm{Diag}(C, \iota)$.

A set $\mathcal{J}$ of computations of a CTS with start state (C, ι) is called *chain-complete* if whenever $\mathcal{I}$ is a subset of $\mathcal{J}$ that is linearly ordered with respect to inclusion, then $\bigcup \mathcal{I}$ is an element of $\mathcal{J}$.

Lemma 2.4.1

1. *The set of computations of a CTS with start state (C, ι) is an algebraic directed-complete poset under inclusion order, whose compact elements are exactly the finite computations.*
2. *Every consistent set of elements of* $\mathrm{Diag}(C, \iota)$ *is included in a unique least computation of (C, ι).*
3. *If $\mathcal{J}$ is a nonempty, chain-complete set of computations of (C, ι), then every element of $\mathcal{J}$ is included in a maximal element of $\mathcal{J}$.*

Proof – (1) That the set of computations of (C, ι) is an algebraic directed-complete poset, with the finite computations as isolated elements, is a standard property of the "completion by ideals" of a partially ordered set (see *e.g.* [11]).

(2) is obvious from the fact that ideals are defined by closure properties.

(3) is just Zorn's Lemma applied to sets of computations. ▮

We conclude this section with a result that shows that our generalized definition of computations as ideals does not depart too radically from the more conventional notion of computation sequences.

A *computation sequence* for a CTS with start state (C, ι) is a sequence $\perp = t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$ of elements of $\mathrm{Diag}(C, \iota)$. The element t_n is called the *result* of the computation sequence. Since every element of $\mathrm{Diag}(C, \iota)$ is the result of some computation sequence for (C, ι) (by the initiality of $\perp$), we may prove properties of finite computations by induction on the length of a computation sequence. A *generalized computation sequence* for (C, ι) is a sequence $\perp = t_0 \preceq t_1 \preceq t_2 \preceq \ldots$ of elements of $\mathrm{Diag}(C, \iota)$, such that for each $k \geq 0$, either $t_k \prec_1 t_{k+1}$ or $t_k = t_{k+1}$. The ideal $\bigvee_{k=0}^{\infty} t_k$ is called the *computation generated by* the sequence $t_0 \preceq t_1 \preceq t_2 \ldots$.

A coinitial set of transitions T of a CTS C is called *independent* if it is consistent, and for all $t \in T$ and all finite sets $T' \subseteq T$, if $t \preceq \bigvee T'$, then $t \in T'$. The CTS C is said to have *finite concurrency* if there are no infinite independent sets of transitions in C.

Theorem 2.6 *Suppose (C, ι) is a CTS with start state, where C has finite concurrency. Then every computation of (C, ι) is generated by some generalized computation sequence of (C, ι).*

Proof – Straightforward. ▮

3 CTS Semantics of Process Networks

In this section, we show how concurrent transition systems can be used as the basis for a dataflow-like model of concurrent computation. The kind of model we consider is similar to those of [16, 17, 6, 10, 24, 34], and concerns a system of processes with internal state that communicate by transmitting messages through named ports. Each port is shared by at most two processes, one of which (called the "receiver") always inputs messages from the port, and the other of which (called the "sender") always outputs messages to the port. Typically, ports are regarded as buffers that transmit messages in FIFO order. However, for our purposes, it will be convenient to take a slightly more abstract point of view in which we think of the state of a port as part of the internal state of the reader of that port. This permits us to regard the transmission of a message by a sender and the arrival of that message at the input port of a receiver as synchronized, simultaneous occurrences. It will also be convenient to adopt a slightly more abstract point of view than usual, regarding the number and type of the ports used by a process. That is, rather than assume that a process has a specific number of input and output ports, each of which is capable of handling values from a certain set, we merely assume that each process has a "type" (X, Y), where X and Y are trace algebras whose elements represent possible message histories for the input and output ports of that process, respectively. The example at the end of Section 3.1 will clarify this point.

Since we will be making frequent use of trace algebras (see Section 2, Example 2), some additional notation will be convenient. If X is a trace algebra, then we denote the identity of X by ϵ_X, or just ϵ, when X is clear from the context. We denote the prefix ordering by $\preceq_X$, or just $\preceq$, and the join by $\vee_X$, or just $\vee$. We identify a trace algebra X with the corresponding one-proper-state CTS, so that trace algebras form a full subcategory of **CTS**. If X and Y are trace algebras, then their product $X \times Y$ in **CTS** is also a trace algebra. We use π_X and π_Y to denote the projections from $X \times Y$ to X and Y, respectively. We use the notation $x; y$ to denote the element of $X \times Y$ with $\pi_X(x; y) = x$ and $\pi_Y(x; y) = y$. If $\lambda : C \to X \times Y$ is a morphism, then we often write λ_X or λ_Y as an abbreviation for $\pi_X \circ \lambda$ and $\pi_Y \circ \lambda$. The *ideal space* of X is the set $\overline{X}$ of ideals of X, which is an algebraic directed-complete poset with respect to inclusion. It will be convenient to regard X as included in $\overline{X}$, by identifying each $x \in X$ with the corresponding principal ideal. We can then view the inclusion order on $\overline{X}$ as an extension of the prefix ordering $\preceq$ on X, and we use the same symbol $\preceq$ in both cases. The elements of $X \subseteq \overline{X}$ are called the *finite* elements of $\overline{X}$, and all other elements of $\overline{X}$ are called *infinite*. A morphism $\rho : X \to Y$ extends uniquely to a continuous function $\overline{\rho} : \overline{X} \to \overline{Y}$. We will generally identify $\overline{X \times Y}$ and $\overline{X} \times \overline{Y}$, exploiting the obvious natural isomorphism.

3.1 Machines

Suppose C is a CTS. An endomorphism $\mu : C \to C$ is *orthogonal* if,

- For all transitions t of C, if $\mu(t)$ is an identity, then t is an identity.
- For all coinitial pairs t, u of transitions of C, if $\mu(t)$ and $\mu(u)$ are consistent, then so are t and u.

An *action* of a trace algebra X on C is a monoid homomorphism $\delta : X \to \mathbf{CTS}(C, C)$, such that for each $x \in X$, the morphism $\delta(x)$ is orthogonal. We usually write δ_x, instead of $\delta(x)$, for the application of an action δ to argument $x \in X$.

Suppose X and Y are trace algebras. An (X, Y)-*machine* is a four-tuple $\mathcal{M} = (C, \iota, \lambda, \delta)$, where:

- C is a CTS, called the *underlying CTS* of $\mathcal{M}$;
- $\iota : \mathbf{1} \to C$ is a morphism, called the *start state map* of $\mathcal{M}$.
- $\lambda : C \to Y$ is a morphism, called the *output map* of $\mathcal{M}$;
- $\delta : X \to \mathbf{CTS}(C, C)$ is an action of X on C, called the *input map* of $\mathcal{M}$;

such that $\lambda \circ \delta_x = \lambda$ for all $x \in X$.

If $\mathcal{M} = (C, \iota, \lambda, \delta)$ and $\mathcal{M}' = (C', \iota', \lambda', \delta')$ are (X, Y)-machines, then a *morphism* from $\mathcal{M}$ to $\mathcal{M}'$ is a morphism $\mu : C \to C'$ of the underlying CTS's, such that

1. $\iota' = \mu \circ \iota$.
2. $\lambda = \lambda' \circ \mu$.
3. $\delta'_x \circ \mu = \mu \circ \delta_x$ for all $x \in X$.

Let $\mathbf{Mach}_{X,Y}$ denote the category of (X, Y)-machines and their morphisms.

A machine is called a *Kahn machine* if its underlying CTS is determinate.

In the special case that $X = \mathbf{1}$ (the terminal object in $\mathbf{CTS}$), an (X, Y)-machine will be called a Y-*automaton.* Note that a Y-automaton is completely specified by giving its underlying CTS C, its start state map ι, and its output map $\lambda : C \to Y$, since there is only one possible input map $\delta : \mathbf{1} \to \mathbf{CTS}(C, C)$. Let $\mathbf{Auto}_Y$ denote the category of Y-automata.

To illustrate the expressive power of the (X, Y)-machine model, we show how the axioms are satisfied by a kind of process that communicates with its environment by reading sequences of values from input ports and writing sequences of values to output ports.

Formally, suppose V is a set of *values*, and m, n are natural numbers. Then a *sequential dataflow process (SDP) with m input ports and n output ports* is a triple (Q, ι, A), where

- Q is a set of *states*, with $\iota \in Q$ a distinguished *start state.*
- $A \subseteq (Q \times (V^*)^m) \times (V^*)^n \times (Q \times (V^*)^m)$ is a set of *transitions.*

such that

1. For all $q \in Q$, the set A contains a transition $((q, \epsilon), \epsilon, (q, \epsilon))$.
2. If the set A contains a transition $((q, x), y, (q', x'))$, then A also contains a transition $((q, xx_0), y, (q', x'x_0))$ for each $x_0 \in X$.

Each transition $((q, x), y, (q', x'))$ in A represents a possible process step, in which a vector x of value sequences on the input ports of the process is replaced by a new vector x' (if $x = x''x'$, then we may think of the prefix x'' of x as being consumed in the step), a vector y of value sequences is transmitted to the output ports of the process, and the internal state of the process is changed from q to q'. We think of input values arriving at the input port of a process as getting concatenated with the current sequence of values in the port. Condition (2) above thus states that arrival of new input values can never cause transitions enabled for a process to become disabled, only new transitions to become enabled.

Any program expressed in a nondeterministic sequential programming language with primitives for reading values from input ports and writing values to output ports, but not for testing for the absence of values on input ports, can be regarded as defining an SDP. For $m \geq 0$, let $(V^*)^m$ be the trace algebra whose elements are m-vectors of elements of the free monoid V^*, with composition and identity defined componentwise. Assuming a suitable definition of the "input/output relation" computed by a process (we shall provide such a definition in the sections to follow), it can be shown that every continuous function from $(\overline{V^*})^m$ to $(\overline{V^*})^n$ is computed by an SDP. An example of a non-functional process also representable as an SDP is "unfair merge," which has two input ports and one output port, and executes a loop in which input is nondeterministically chosen from one of the input ports and output on the output port. This merge is "unfair" because we do not make any assumption about how often in a computation one enabled branch of a nondeterministic choice may be rejected in favor of another (although we do introduce a kind of fairness assumption with respect to choices between *consistent* transitions).

An SDP (Q, ι, A) may be regarded as an (X, Y)-machine $\mathcal{M} = (C, \iota, \lambda, \delta)$ as follows:

- Let X be the trace algebra $(V^*)^m$ and Y the trace algebra $(V^*)^n$.
- Let the CTS C have as proper states the elements of Q, and as proper transitions the elements of A. Define

$$\begin{aligned} \mathrm{dom}((q, x), y, (q', x')) &= q \\ \mathrm{cod}((q, x), y, (q', x')) &= q'. \end{aligned}$$

  Let the identities of C be the transitions $((q, \epsilon), \epsilon, (q, \epsilon))$, and let $\uparrow$ be defined so that two coinitial transitions are consistent iff they are equal, or one is an identity.
- Let $\lambda : C \to Y$ take $((q, x), y, (q', x'))$ to y.
- Let $\delta : X \to \mathbf{CTS}_\iota(C, C)$ be defined so that δ_{x_0} takes a transition $((q, x), y, (q', x')) \in A$ to the transition $((q, xx_0), y, (q', x'x_0)) \in A$, which exists by condition (2) in the definition of an SDP above.

It is straightforward to check that these definitions satisfy the requirements for an (X, Y)-machine.

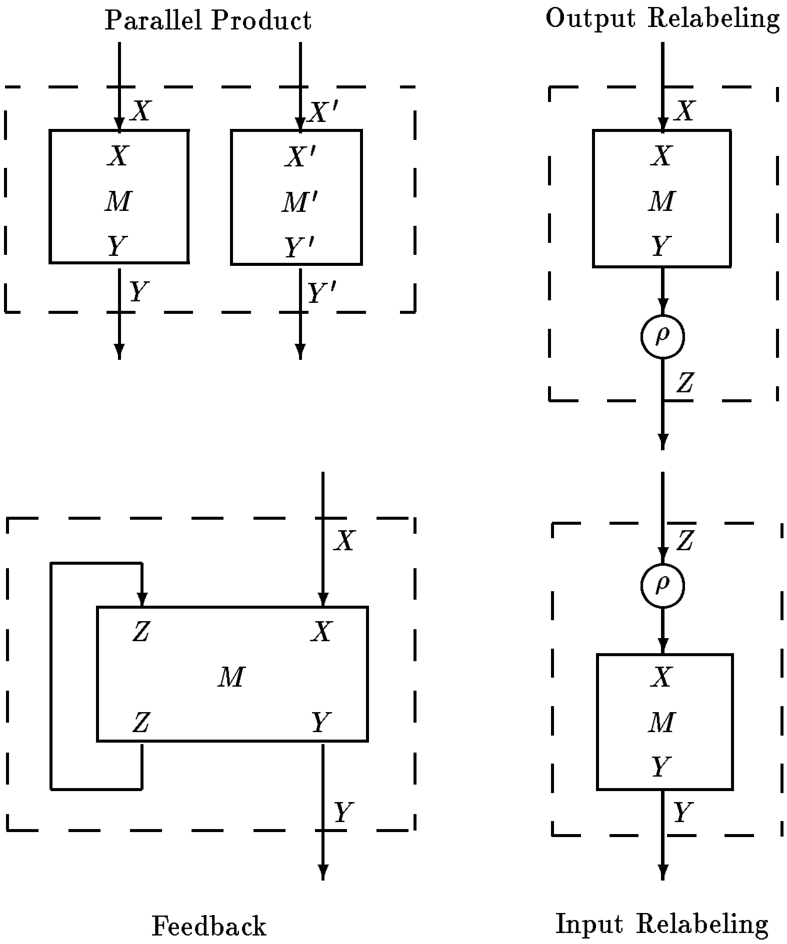

Figure 4: Operations on Machines

3.2 An Algebra of Machines

We are interested in the properties of an algebra of machines, with respect to operations that correspond to ways of building more complex machines from simpler components. Although many such operations can be defined, in this paper we restrict our attention to *parallel product*, input and output *relabeling*, and *feedback*. The effect of these operations is depicted schematically in Figure 4.

3.2.1 Parallel Product

Suppose $\mathcal{M} = (C, \iota, \lambda, \delta)$ is an (X, Y)-machine and $\mathcal{M}' = (C', \iota', \lambda', \delta')$ is an (X', Y')-machine. Define the *parallel product* of $\mathcal{M}$ and $\mathcal{M}'$ to be the $(X \times X', Y' \times Y)$-machine

$$\mathcal{M} \times \mathcal{M}' = (C \times C', (\iota, \iota'), \lambda'', \delta''),$$

where λ'' takes each transition (t, t') of $\mathcal{M} \times \mathcal{M}'$ to the trace $\lambda'(t'); \lambda(t) \in Y' \times Y$, and $\delta''_{x;x'}(t, t') = (\delta_x(t), \delta'_x(t'))$ for each $x; x' \in X \times X'$ and transition (t, t') of $C \times C'$.

Parallel product is easily seen to be the object map of a functor from $\mathbf{Mach}_{X,Y} \times \mathbf{Mach}_{X',Y'}$ to $\mathbf{Mach}_{X\times X',Y'\times Y}$.

3.2.2 Output Relabeling

Suppose $\mathcal{M} = (C, \iota, \lambda, \delta)$ is an (X, Y)-machine, and $\rho : Y \to Z$ is a morphism. Define the *output relabeling* of $\mathcal{M}$ by ρ to be the (X, Z)-machine

$$\mathcal{M} \rhd \rho = (C, \iota, \rho \circ \lambda, \delta).$$

Output relabeling is the object map of a functor from $\mathbf{Mach}_{X,Y}$ to $\mathbf{Mach}_{X,Z}$.

3.2.3 Input Relabeling

Suppose $\mathcal{M} = (C, \iota, \lambda, \delta)$ is an (X, Y)-machine, and $\rho : Z \to X$ is a morphism. Define the *input relabeling* of $\mathcal{M}$ by ρ to be the (Z, Y)-machine

$$\rho \rhd \mathcal{M} = (C, \iota, \lambda, \delta \circ \rho).$$

Input relabeling is the object map of a functor from $\mathbf{Mach}_{X,Y}$ to $\mathbf{Mach}_{Z,Y}$.

3.2.4 Feedback

Suppose $\mathcal{M} = (C, \iota, \lambda, \delta)$ is a $(Z \times X, Y \times Z)$-machine. Define the *feedback* $\{\mathcal{M}\}_{\circlearrowleft Z}$ of $\mathcal{M}$ with respect to Z to be the $(X, Y \times Z)$-machine

$$\{\mathcal{M}\}_{\circlearrowleft Z} = (C', \iota, \lambda', \delta'),$$

where

- C' is a CTS whose states and transitions are the same as those of C, but whose domain and codomain functions are defined as follows:

$$\begin{aligned} \mathrm{dom}'(t) &= \mathrm{dom}(t) \\ \mathrm{cod}'(t) &= \delta_{\lambda_Z(t);\epsilon_X}(\mathrm{cod}(t)) \end{aligned}$$

  The identities of C' are the same as those of C. For coinitial t, u, define

$$t \uparrow' u = \delta_{\lambda_Z(u);\epsilon_X}(t \uparrow u).$$

- $\lambda' : C' \to Y$ is defined by $\lambda'(t) = \lambda(t)$.
- $\delta' : X \to \mathbf{CTS}(C', C')$ is defined by $\delta'_x(t) = \delta_{\epsilon_Z;x}(t)$.

It is straightforward to verify that $\{\mathcal{M}\}_{\circlearrowleft Z}$ is, in fact, an $(X, Y \times Z)$-machine. The assumption that $\delta_{z;x}$ is orthogonal for all $z; x \in Z \times X$ is used in the verification of CTS axioms (3) and (4).

Intuitively, the machine $\{\mathcal{M}\}_{\circlearrowleft Z}$ represents the machine $\mathcal{M}$ with its Z-output "fed back" to its Z-input. Thus, each transition t of $\{\mathcal{M}\}_{\circlearrowleft Z}$ is a transition of $\mathcal{M}$ that has been "composed" with the effect, given by $\delta_{\lambda_Z(t);\epsilon_X}$, of the feedback input generated by t.

The feedback operation can be shown to be the object map of a functor from $\mathbf{Mach}_{Z\times X,Y\times Z}$ to $\mathbf{Mach}_{X,Y\times Z}$.

3.3 Observable Equivalence

Define the *pairing* of an (X, Y) machine $\mathcal{M}$ and a (Y, X)-machine $\mathcal{N}$ to be the $(X \times Y)$-automaton

$$\langle \mathcal{M}, \mathcal{N} \rangle = \{\mathcal{M} \times \mathcal{N}\}_{\circlearrowleft X \times Y}.$$

Intuitively, the automaton $\langle \mathcal{M}, \mathcal{N} \rangle$ represents a closed system consisting of machines $\mathcal{M}$ and $\mathcal{N}$ executing in parallel, with the output of $\mathcal{M}$ feeding the input of $\mathcal{N}$ and the output of $\mathcal{N}$ feeding the input of $\mathcal{N}$.

Lemma 3.3.1

1. *Suppose $\mathcal{M}$ is an (X, Y)-machine, $\mathcal{N}$ is a (Y, X)-machine, and $\pi : X \times Y \to Y \times X$ takes $x; y$ to $y; x$. Then*
$$\langle \mathcal{M}, \mathcal{N} \rangle \simeq \langle \mathcal{N}, \mathcal{M} \rangle \rhd \pi.$$

2. *Suppose $\mathcal{M}$ is an (X, Y)-machine, $\mathcal{N}$ is a (Z, X)-machine, and $\rho : Y \to Z$. Then*
$$\langle \mathcal{M} \rhd \rho, \mathcal{N} \rangle \simeq \langle \mathcal{M}, \rho \rhd \mathcal{N} \rangle \rhd (\pi_X \times (\rho \circ \pi_Y)).$$

3. *Suppose $\mathcal{M}$ is an (X, Y)-machine, $\mathcal{N}$ is a (Z, W)-machine, and $\mathcal{P}$ is a $(W \times Y, X \times Z)$-machine. Then*
$$\langle \mathcal{M} \times \mathcal{N}, \mathcal{P} \rangle \simeq \langle \mathcal{M}, \{\mathcal{N} \times \mathcal{P}\}_{\circlearrowleft Z \times W} \rangle.$$

Proof – Straightforward. ∎

Although a result exactly analogous to (2) and (3) above does not hold for the feedback operation, we can obtain a useful weaker result (Lemma 3.3.2 below). To state it, we need to define the *output set* of an automaton. Output sets of automata are a special cases of *input/output relations* of machines, which we will define and investigate in more detail in Section 4.

Suppose $\mathcal{A} = (C, \iota, \lambda)$ is a Y-automaton. The *computations of* $\mathcal{A}$ are the computations of the CTS with start state (C, ι). A computation J of $\mathcal{A}$ is *maximal* if it is not a proper subset of any other computation of $\mathcal{A}$. Now, if $\pi_C : \mathrm{Diag}(C, \iota) \to (C, \iota)$ denotes the universal morphism associated with the right adjoint Diag, then $\lambda \circ \pi_C : \mathrm{Diag}(C, \iota) \to Y$ extends uniquely to a continuous map $\overline{\lambda}$ from the cpo of computations of (C, ι) to the ideal space $\overline{Y}$ of Y. Thus, each computation J of $\mathcal{A}$ determines an ideal $\overline{\lambda}(J) \in \overline{Y}$, which we call the *complete output trace* of J. The *output set* $\mathrm{Out}(\mathcal{A})$ of $\mathcal{A}$ is the set of all complete output traces determined by maximal computations of $\mathcal{A}$.

We also need a (Z, Z)-machine $\mathcal{I}_Z$ that simply passes its input through unchanged to its output. Lemma 4.6.2 provides such a machine, and we anticipate this result here, rather than restating a special case.

Lemma 3.3.2 *Suppose $\mathcal{M}$ is a $(Z \times X, Y \times Z)$-machine, and $\mathcal{N}$ is a $(Y \times Z, X)$-machine. Let $\rho : Y \times Z \to Y \times Z \times Z$ be the morphism that takes $y; z$ to $y; z; z$, and let $\pi : Z \times X \times Y \times Z \to X \times Y \times Z$ be the morphism that takes $z'; x; y; z$ to $x; y; z$. Then*

$$\mathrm{Out}\langle \{\mathcal{M}\}_{\circlearrowleft Z}, \mathcal{N} \rangle = \mathrm{Out}(\langle \mathcal{M}, \rho \rhd (\mathcal{N} \times \mathcal{I}_Z) \rangle \rhd \pi).$$

Proof – Suppose $\langle \{\mathcal{M}\}_{\circlearrowleft Z}, \mathcal{N} \rangle = (C, \iota, \lambda)$ and $\langle \mathcal{M}, \rho \rhd (\mathcal{N} \times \mathcal{I}_Z) \rangle \rhd \pi = (C', \iota', \lambda')$. Let $(D, \lambda) = \mathrm{Diag}(C, \iota)$ and $(D', \lambda') = \mathrm{Diag}(C', \iota')$. We can construct morphisms $\rho : D^* \to (D')^*$ and $\rho' : (D')^* \to D^*$, preserving the initial state, and such that $(\lambda')^* = \lambda^* \circ \rho$, $\rho' \circ \rho = \mathrm{id}_{\mathrm{Diag}(C,\iota)}$, and $t' \preceq \rho(\rho'(t'))$ for all elements t' of $\mathrm{Diag}(C', \iota)$. From this, it follows that the output sets of the two automata are identical. We omit the details. ∎

We say that (X, Y)-machines $\mathcal{M}$ and $\mathcal{M}'$ are *observably equivalent*, and we write $\mathcal{M} \sim \mathcal{M}'$, if we have

$$\mathrm{Out}\langle \mathcal{M}, \mathcal{N} \rangle = \mathrm{Out}\langle \mathcal{M}', \mathcal{N} \rangle$$

for all (Y, X)-machines $\mathcal{N}$. Here we use the idea of defining process equivalences based on indistinguishability with respect to tests performed by "observers" or "environments," which is discussed in [12].

Theorem 3.1 *Observable equivalence is a congruence with respect to parallel product, output relabeling, and feedback. It is also a congruence with respect to input relabeling by left-invertible morphisms. That is,*

1. *Suppose $\mathcal{M}$ and $\mathcal{M}'$ are (X, Y)-machines, and $\mathcal{M} \sim \mathcal{M}'$. Then*
   (a) *$\mathcal{M} \times \mathcal{N} \sim \mathcal{M}' \times \mathcal{N}$, for all machines $\mathcal{N}$.*
   (b) *$\mathcal{M} \rhd \rho \sim \mathcal{M}' \rhd \rho$ for all morphisms $\rho : Y \to Z$.*
   (c) *$\rho \rhd \mathcal{M} \sim \rho \rhd \mathcal{M}'$ for all morphisms $\rho : Z \to X$ for which there exists a morphism $\rho' : X \to Z$ with $\rho' \circ \rho = \mathrm{id}_Z$.*
2. *Suppose $\mathcal{M}$ and $\mathcal{M}'$ are $(Z \times X, Y \times Z)$-machines. If $\mathcal{M} \sim \mathcal{M}'$, then $\{\mathcal{M}\}_{\circlearrowleft Z} \sim \{\mathcal{M}'\}_{\circlearrowleft Z}$.*

Moreover, $\sim$ is the largest congruence on machines, respecting parallel product and feedback, that does not relate automata with distinct output sets.

Proof – (1a) Let $\mathcal{P}$ be an arbitrary $(W \times Y, X \times Z)$-machine, and let π be as in Lemma 3.3.1(3). Using that lemma and the equivalence of $\mathcal{M}$ and $\mathcal{M}'$, we have:

$$\begin{aligned} \mathrm{Out}\langle \mathcal{M} \times \mathcal{N}, \mathcal{P} \rangle &= \mathrm{Out}\langle \mathcal{M}, \{\mathcal{N} \times \mathcal{P}\}_{\circlearrowleft Z \times W} \rangle \\ &= \mathrm{Out}\langle \mathcal{M}', \{\mathcal{N} \times \mathcal{P}\}_{\circlearrowleft Z \times W} \rangle \\ &= \mathrm{Out}\langle \mathcal{M}' \times \mathcal{N}, \mathcal{P} \rangle. \end{aligned}$$

(1b) Similar to (1a), but using Lemma 3.3.1(2).
(1c) Similar to (1b), but using the isomorphism

$$\langle \rho \rhd \mathcal{M}, \mathcal{N} \rangle \simeq \langle \mathcal{M}, \mathcal{N} \rhd \rho \rangle \rhd ((\rho' \circ \pi_X) \times \pi_Y),$$

obtained from Lemma 3.3.1(1, 2), plus the assumption that $\rho' \circ \rho = \mathrm{id}_Z$.

(2) Similar to the above, but using Lemma 3.3.2.

To show that $\sim$ is the largest congruence that respects parallel product and feedback, but does not relate automata with distinct output sets, suppose $\approx$ is a congruence on machines that relates two (X, Y)-machines $\mathcal{M}$ and $\mathcal{M}'$ that are not related by $\sim$. By definition of $\sim$, there exists a (Y, X)-machine $\mathcal{N}$ such that

$$\mathrm{Out}\langle \mathcal{M}, \mathcal{N} \rangle \neq \mathrm{Out}\langle \mathcal{M}', \mathcal{N} \rangle.$$

But we must have $\langle \mathcal{M}, \mathcal{N} \rangle \approx \langle \mathcal{M}', \mathcal{N} \rangle$ by the assumption that $\approx$ is a congruence with respect to parallel product and feedback. Hence $\approx$ relates two automata with distinct output sets. ∎

The *full abstraction problem for machines* is the problem of characterizing the structure of the quotient algebra of machines modulo observable equivalence. The difficulty of this problem has become apparent since it was first pointed out by Keller [18] and more conclusively by Brock and Ackerman [7] (the so-called "Brock-Ackerman anomaly") that the mapping taking machines to their input/output relations is not homomorphic with respect to feedback. Since then, a number of researchers [31, 32, 3, 19, 34, 20] have proposed process models that incorporate somewhat more information than just input/output relations. Abramsky [1], has shown the full abstractness of a powerdomain model, with respect to a notion of observable equivalence based on finite computations. To the author's knowledge, though, none of these models has been shown both consistent with (*i.e.* a homomorphic image of) an operational semantics as well as fully abstract (*i.e.* observably equivalent processes have identical images), when both infinite and finite computations are considered.

4 An Analysis of Feedback

It would seem that a deep understanding of the feedback operation is prerequisite to solving the full abstraction problem. Whereas the parallel product operation is readily seen to be respected by the mapping from machines to input/output relations, the same does not hold for the feedback operation. The input/output relation of a $(Z \times X, Y \times Z)$-machine $\mathcal{M}$ simply does not contain enough information, in general, to determine the input/output relation of $\{\mathcal{M}\}_{\circlearrowleft Z}$.

In an attempt to make progress on the full abstraction problem, in this section we investigate how the feedback operation behaves under a sequence of mappings that starts with machines and ends with input/output relations. The idea is to try to delete more and more information, getting successively more abstract representations of the behavior of machines, until we cannot see how to delete any more information and still preserve the machine operations. Some of the representations at intermediate stages between machines and input/output relations are similar to various models that have been proposed for concurrent processes. Thus, as a byproduct of our analysis, we obtain an improved understanding of the relationship between these models.

Our first mapping takes an (X, Y)-machine to a corresponding $(X \times Y)$-automaton. Whereas machines can be thought of as a CTS generalization of the sequential machines of classical automata theory, automata can be thought of as a CTS generalization of classical nondeterministic automata, or the labeled transition systems used, *e.g.* in [9, 8]. Our second mapping "unwinds" automata to obtain their "synchronization diagrams," which are a generalization of "synchronization trees" [42]. We then show how each synchronization diagram determines a set of "behaviors," which represent "fair" or "completed" computations. Abstracting further from sets of behaviors, we obtain sets of "histories," which are related to the "pomset" model of [32, 33]. We then map sets of histories to sets of "scenarios", where a scenario represents information about causal relationships between input and output in a single computation. Our scenarios are similar in spirit, although not formally identical to, the scenarios originally defined by Brock and Ackerman [7, 6]. Finally, we show how scenario sets determine input/output relations.

For each of the mappings, we prove a theorem showing a sense in which the mapping is homomorphic with respect to the feedback operation on machines. All of the mappings except the one to input/output relations are homomorphic with respect to the full algebra of machines. The mapping from to input/output relations is homomorphic only on the subalgebra of Kahn machines. (Its failure to be homomorphic for unrestricted machines is the Brock-Ackerman anomaly already mentioned.) We show that the input/output relation of an (X,Y)-Kahn machine is the graph of a continuous function from $\overline{X}$ to $\overline{Y}$, and that the map from Kahn machines to continuous functions transforms feedback into a certain least-fixed-point construction. This least-fixed-point characterization of feedback was first noted by Kahn [16], and has been called the "Kahn Principle." Although the Kahn Principle has been proved before [10], our proof applies to a more general, axiomatically defined class of processes.

4.1 Automata

Given $\overline{x} \in \overline{X}$, let $\hat{X}$ denote the (Y,X)-machine $(X, \iota, \mathrm{id}_X, \delta)$, where ι is the unique proper state of X and $\delta_y : X \to X$ is id_X for each $y \in Y$. Intuitively, $\hat{X}$ is a machine that ignores its input, and is capable of outputting an arbitrary element of X at any time. Each (X,Y)-machine $\mathcal{M}$ determines a corresponding $(X \times Y)$-automaton $\mathrm{Auto}(\mathcal{M})$, under the definition

$$\mathrm{Auto}(\mathcal{M}) = \langle \mathcal{M}, \hat{X} \rangle.$$

Define an $(X \times Y)$-automaton $\mathcal{A}$ to be an (X,Y)-*input/output automaton* (resp. (X,Y)-*Kahn automaton*) if $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$ for some (X,Y)-machine (resp. (X,Y)-Kahn machine) $\mathcal{M}$.

We now characterize the structure of input/output automata. To state this result, some additional terminology will be convenient. Suppose C is a CTS, and $\rho : C \to X$ is a morphism. We say that a proper transition t of $\mathcal{A}$ is *canonical* w.r.t. ρ, if for any other proper transition t' of C, with $\mathrm{dom}(t) = \mathrm{dom}(t')$ and $\rho(t') = \rho(t)$, we have $t \preceq t'$. Note that canonical transitions, when they exist, are uniquely determined by their domain and their image under ρ.

Theorem 4.1 *An $(X \times Y)$-automaton $\mathcal{A} = (C, \iota, \lambda)$ is an (X,Y)-input/output automaton iff $\mathcal{A}$ has the following properties:*

1. *For each proper state q of $\mathcal{A}$, and each $x \in X$, there exists a transition x_q of C, with $\mathrm{dom}(x_q) = q$ and $\lambda_X(x_q) = x$, such that x_q is canonical w.r.t. λ_X.*
2. *Each proper transition $t : q \to r$ of $\mathcal{A}$, has a decomposition $t = x_q \vee u$, where $x = \lambda_X(t)$.*
3. *For each proper transition $t : q \to r$ of $\mathcal{A}$, and each $x \in X$, we have $x_q \uparrow t = (x \uparrow \lambda(t))_r$.*
4. *For each proper transition $t : q \to r$ of $\mathcal{A}$, and each $x \in X$, if x and $\lambda(t)$ are consistent, then x_q and t are consistent, and $x_q \vee t$ exists in $\mathcal{A}$.*
5. *For each proper transition $t : q \to r$ of $\mathcal{A}$, and each $x \in X$, if $\lambda_X(t) = \epsilon$ and $t \uparrow x_q = \mathrm{id}_r$, then $t = \mathrm{id}_q$.*

Moreover, $\mathcal{A}$ is a Kahn automaton iff it has the additional property:

6. *For each coinitial pair t, u of proper transitions of $\mathcal{A}$, if $\lambda_X(t)$ and $\lambda_X(u)$ are consistent, then t and u are consistent.*

Proof – We show: ($\Rightarrow$) If $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$ for some $\mathcal{M}$, then $\mathcal{A}$ has properties (1)-(5), and also property (6) in case $\mathcal{M}$ is a Kahn machine. ($\Leftarrow$) If $\mathcal{A}$ has properties (1)-(5), then $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$ for some $\mathcal{M}$, and if $\mathcal{A}$ has property (6), then $\mathcal{M}$ can be chosen to be a Kahn machine.

($\Rightarrow$) Suppose $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$. Since properties (1)-(5) are preserved under isomorphism, we may suppose without loss of generality that $\mathcal{A} = \mathrm{Auto}(\mathcal{M})$. Then the state set of $\mathcal{A}$ is (in bijective correspondence with) the state set of $\mathcal{M}$, and the transitions of $\mathcal{A}$ are pairs (t, x), where t is a transition of $\mathcal{M}$ and $x \in X$. It is easily shown that the transitions $x_q = (\mathrm{id}_q, x)$ are the canonical transitions required by property (1). The remaining properties then follow by straightforward calculations, which we omit.

($\Leftarrow$) Suppose $\mathcal{A} = (C, \iota, \lambda)$ has properties (1)-(5). We show how to construct $\mathcal{M}$ so that $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$. It is easily verified that the states of C, equipped with all transitions t of C such that $\lambda_X(t) = \epsilon$, define a sub-automaton $\mathcal{A}' = (C', \iota, \lambda')$ of $\mathcal{A}$. Let $\delta : X \to \mathbf{CTS}(C', C')$ be defined by $\delta_x(t) = t \uparrow x_{\mathrm{dom}(q)}$. Straightforward calculations, which we omit, from the definitions and properties (1)-(5) suffice to verify that $\mathcal{M} = (C', \iota, \lambda'_Y, \delta)$ is an (X, Y)-machine. It is also easily verified that if $\mathcal{A}$ has property (6), then $\mathcal{M}$ is a Kahn machine.

We claim that $\mathcal{A} \simeq \mathrm{Auto}(\mathcal{M})$. The required isomorphism $\mu : \mathrm{Auto}(\mathcal{M}) \to \mathcal{A}$ is obtained as the morphism that takes each transition $(u, x) : \mathrm{dom}(u) \to \delta_x(\mathrm{cod}(u))$ of $\mathrm{Auto}(\mathcal{M})$ to the join $x_q \vee u$ in $\mathcal{A}$, which exists by property (4). The morphism μ is surjective on transitions because by property (2), every proper transition t of $\mathcal{A}$ has a decomposition $t = x_q \vee u$, where $x = \lambda_X(t)$. To show that it is injective on transitions, it suffices to show the uniqueness of such decompositions in $\mathcal{A}$. If $x_q \vee u$ and $x_q \vee u'$ are two decompositions of t, then $(x_q \vee u) \uparrow (x_q \vee u')$ is an identity transition. Since $(x_q \vee u) \uparrow (x_q \vee u') = [x_q \uparrow (x_q \vee u')] \vee [u \uparrow (x_q \vee u')]$, it follows that $u \uparrow (x_q \vee u')$ is an identity transition. However, $u \uparrow (x_q \vee u') = (u \uparrow u') \uparrow (x_q \uparrow u')$, and $x_q \uparrow u' = x_{\mathrm{cod}(u')}$ by property (3). Hence $u \uparrow u'$ is an identity transition by property (5). Similar reasoning shows that $u' \uparrow u$ is an identity transition, thus $u = u'$. ∎

In case $\mathcal{A} = (C, \iota, \lambda)$ is an (X, Y)-input/output automaton, we will refer to the canonical transitions x_q of $\mathcal{A}$ as *pure-input* transitions, and to decompositions $t = x_q \vee v$, with $x = \lambda_X(t)$, as *pure-input/output* decompositions. It will also sometimes be convenient to use λ^{in} and λ^{out} as alternate notations for λ_X and λ_Y, respectively.

We now determine how the feedback operation on machines is transformed by the map from machines to automata. The relabeling functor

$$\text{-} \rhd (\pi_Z \times \mathrm{id}_{W \times Z}) : \mathbf{Auto}_{W \times Z} \to \mathbf{Auto}_{Z \times W \times Z}$$

has a right adjoint

$$\{\text{-}\}_{=Z} : \mathbf{Auto}_{Z \times W \times Z} \to \mathbf{Auto}_{W \times Z},$$

whose object map takes each $Z \times W \times Z$-automaton $\mathcal{A} = (C, \iota, \lambda)$ to the $(W \times Z)$-automaton $\{\mathcal{A}\}_{=Z} = (C', \iota, \lambda')$, where C' is the sub-CTS of C consisting of all transitions t of C for which the two Z components of $\lambda(t)$ are equal, and λ' takes each t in C' to $\lambda_{W \times Z}(t)$.

Theorem 4.2 *Suppose $\mathcal{M}$ is a $(Z \times X, Y \times Z)$-machine. Then*

$$\mathrm{Auto}(\{\mathcal{M}\}_{\circlearrowleft Z}) \simeq \{\mathrm{Auto}(\mathcal{M})\}_{=Z}$$

Proof – If $\mathcal{M} = (C, \iota, \lambda, \delta)$, then the required isomorphism maps a transition (t, x) of $\mathrm{Auto}(\{\mathcal{M}\}_{\circlearrowleft Z})$ to the corresponding transition $(t, \lambda_Z(t); x)$ of $\{\mathrm{Auto}(\mathcal{M})\}_{=Z}$. The details are straightforward, and are omitted. ∎

4.2 Synchronization Diagrams

If W is a trace algebra, then a *W-synchronization diagram* is a W-automaton $\mathcal{D} = (D, \perp, \lambda)$, whose underlying CTS D is a computation diagram with initial state $\perp$. Since $\perp$ can be determined from the structure of D, it is redundant information for synchronization diagrams, and we henceforth omit mention of it. Let $\mathbf{Diag}_W$ denote the full subcategory of $\mathbf{Auto}_W$, whose objects are the W-synchronization diagrams.

Suppose $\mathcal{A} = (C, \iota, \lambda)$ is a W-automaton. Define the *diagram of $\mathcal{A}$* to be the synchronization diagram $\mathrm{Diag}(\mathcal{A}) = (\mathrm{Diag}(C, \iota), \lambda \circ \pi_C)$, where $\pi_C : \mathrm{Diag}(C, \iota) \to (C, \iota)$ is the universal morphism associated with the right adjoint Diag.

An $(X \times Y)$-synchronization diagram is called an *(X, Y)-input/output diagram* (resp. (X, Y)-Kahn diagram) if it is isomorphic to $\mathrm{Diag}(\mathcal{A})$ for some (X, Y)-input/output automaton (resp. (X, Y)-Kahn automaton) $\mathcal{A}$.

Lemma 4.2.1 *An $(X \times Y)$-synchronization diagram is an (X, Y)-input/output diagram (resp. (X, Y)-Kahn diagram) iff it is an (X, Y)-input/output automaton (resp. (X, Y)-Kahn automaton).*

Proof – The result is easily established by noting that the properties of Theorem 4.1 are preserved under the "unwinding construction" by which $\mathrm{Diag}(\mathcal{A})$ is obtained from $\mathcal{A}$. ▮

Lemma 4.2.2 *Suppose $\mathcal{D} = (D, \lambda)$ is an (X, Y)-input/output diagram.*

1. *For all $x \in X$, there exists an element t_x of D such that $\lambda_X(t_x) = x$, and such that if u is any element of D with $\lambda_X(u) = \lambda_X(t_x)$, then $t_x \preceq u$.*

2. *If u is an element of D, and $x \in X$ is consistent with $\lambda_X(u)$, then t_x and u are consistent.*

3. *Suppose $\mathcal{D}$ is a Kahn diagram. If t and u are two elements of D such that $\lambda_X(t)$ and $\lambda_X(u)$ are consistent, then t and u are consistent.*

Proof – Straightforward from Lemma 4.2.1. ▮

The elements t_x in (1) above are called the *pure-input elements* of $\mathcal{D}$.

We now determine the form taken by the feedback operation on synchronization diagrams. Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times W \times Z)$-synchronization diagram. Let λ_{Z_1} and λ_{Z_2} be the projections of λ on the first and second Z components, respectively. Define a *feedback computation sequence* for $\mathcal{D}$ to be a computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$ for $\mathcal{D}$, such that $\lambda_{Z_1}(t_k) = \lambda_{Z_2}(t_k)$ for $1 \leq k \leq n$. An element of $\mathcal{D}$ is called *feedback-reachable* if it is the result of some feedback computation sequence for $\mathcal{D}$, and a state of $\mathcal{D}$ is called *feedback-reachable* if it is the codomain of some feedback-reachable element of $\mathcal{D}$. The *feedback-reachable subdiagram* of D is the full subdiagram D' of D whose states are all the feedback-reachable states of D. Define $\{\mathcal{D}\}_{\doteq Z} = (D', \lambda')$, where D' is the feedback-reachable subdiagram of D, and $\lambda'(t) = \lambda_{W \times Z}(t)$.

Theorem 4.3 *Suppose $\mathcal{A}$ is a $(Z \times W \times Z)$-automaton. Then*

$$\mathrm{Diag}(\{\mathcal{A}\}_{=Z}) \simeq \{\mathrm{Diag}(\mathcal{A})\}_{\doteq Z}.$$

Proof – Straightforward from the observation that both functors

$$\mathrm{Diag}(\{\text{-}\}_{=Z}) : \mathbf{Auto}_{Z\times W\times Z} \to \mathbf{Diag}_{W\times Z}$$

and

$$\{\mathrm{Diag}(\text{-})\}_{\dot{=}Z} : \mathbf{Auto}_{Z\times W\times Z} \to \mathbf{Diag}_{W\times Z}$$

are right-adjoint to the composition of the output-relabeling functor

$$\text{-} \rhd (\pi_Z \times \mathrm{id}_{W\times Z}) : \mathbf{Auto}_{W\times Z} \to \mathbf{Auto}_{Z\times W\times Z}$$

with the inclusion of $\mathbf{Diag}_{W\times Z}$ in $\mathbf{Auto}_{W\times Z}$. Since two right adjoints to the same functor are naturally isomorphic, the result follows. ▮

4.3 Behaviors

If $\mathcal{D} = (D, \lambda)$ is a W-synchronization diagram, then each consistent set J of elements of $\mathcal{D}$ determines a consistent subset $\lambda(J)$ of W, which we call the *history* of J with respect to λ. The set $\lambda(J)$ extends to a least ideal $\overline{\lambda}(J) \in \overline{W}$, which we call the *complete trace* of J with respect to λ. If $\mathcal{D}$ is an (X, Y)-input/output diagram, and $\overline{\lambda}(J) = \overline{x};\overline{y}$, then we call $\overline{x}$ the *complete input trace*, and $\overline{y}$ the *complete output trace*, of J. A *behavior* of $\mathcal{D}$ is a computation J of D that is maximal among all computations of D with the same complete input trace as J.

Lemma 4.3.1 *Suppose $\mathcal{D} = (D, \lambda)$ is an (X, Y)-input/output diagram. Then*

1. *Each consistent set J of elements of D extends to a behavior of $\mathcal{D}$ having the same complete input trace as J. In particular, for each $\overline{x} \in \overline{X}$ there is a behavior of $\mathcal{D}$ with $\overline{x}$ as its complete input trace. Moreover, if $\mathcal{D}$ is a Kahn diagram, then its behaviors are uniquely determined by their complete input traces.*

2. *If J is a behavior of $\mathcal{D}$ with complete input trace $\overline{x}$, and $\overline{x} \preceq \overline{x}'$, then J extends to a behavior J' of $\mathcal{D}$ with complete input trace $\overline{x}'$.*

3. *If $\{J_i : i \in I\}$ is any directed collection of behaviors of $\mathcal{D}$, then $\bigvee\{J_i : i \in I\}$ is also a behavior of $\mathcal{D}$.*

Proof – (1) Suppose J is a consistent set of elements of D. Then the class of all consistent sets of elements of D with the same complete input trace as J is nonempty, and is easily seen to be chain-complete. Hence by Lemma 2.4.1 this class has a maximal element, which is a behavior of $\mathcal{D}$ with the same complete input trace as J. In the special case that the given set J is the set of all pure-input elements of D whose input traces are prefixes of $\overline{x}$, this construction yields a behavior of $\mathcal{D}$ with complete input trace $\overline{x}$. Moreover, the existence of two distinct behaviors of $\mathcal{D}$ with the same complete input trace implies the existence of two inconsistent elements of D with consistent input traces. Since by Lemma 4.2.2, this cannot happen when $\mathcal{D}$ is a Kahn diagram, we conclude that behaviors of Kahn diagrams are uniquely determined by their input traces.

(2) Given a behavior J of $\mathcal{D}$ with complete input trace $\overline{x}$, and given $\overline{x}'$ with $\overline{x} \preceq \overline{x}'$, let K be the set of all pure-input elements of D whose input traces are prefixes of $\overline{x}'$. Then the set $J \cup K$ is

consistent by Lemma 4.2.2, and has complete input trace $\overline{x}'$, hence it extends to a behavior J' of $\mathcal{D}$ that has complete input trace $\overline{x}'$.

(3) Suppose the set $J = \bigvee\{J_i : i \in I\}$ were not a behavior of $\mathcal{D}$. Then there would exist some element t of D whose input trace is a prefix of the complete input trace of J, but such that $t \notin J$. But some J_i must have a complete input trace with that of t as a prefix, hence $t \in J_i$ because J_i is a behavior of $\mathcal{D}$. Since this contradicts the assumption $t \notin J$, we conclude that t cannot exist. ∎

If $(K, \preceq)$ is a partially ordered set, and $J \subseteq K$, then J is called *cofinal in K* if for every element t of K there exists an element u of J with $t \preceq u$.

Lemma 4.3.2 *Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times W \times Z)$-synchronization diagram, and K is a computation of D. Let J be the set of feedback-reachable elements of K. Then J is cofinal in K iff J and K have the same complete trace.*

Proof – If J is cofinal in K then it is obvious that J and K have the same complete trace. Conversely, suppose K has complete trace $\overline{z}; \overline{w}; \overline{z}$. Let $t \in K$ be given; then we can choose $u \in J$ such that $\lambda(t) \preceq \lambda(u)$. Since K is determinate, t and u must be consistent, and hence $\lambda(t \uparrow u) = \epsilon$. Since $u \in J$, hence is feedback-reachable in K, it then follows easily that $t \vee u = u(t \uparrow u)$ is feedback-reachable in K, hence is in J. Thus, $t \vee u \in J$ is such that $t \preceq t \vee u$, as required. ∎

Theorem 4.4 *Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times X, Y \times Z)$-input/output diagram. Then a set J of elements of D is a behavior of $\{\mathcal{D}\}_{\doteq Z}$ iff there exists a behavior K of $\mathcal{D}$, such that J is the set of feedback-reachable elements of K, and J is cofinal in K.*

Proof – $(\Rightarrow)$ Suppose J is a behavior of $\{D\}_{\doteq Z}$, with $\overline{\lambda}(J) = \overline{z}; \overline{x}; \overline{y}; \overline{z}$. By Lemma 4.3.1, J extends to a behavior K of $\mathcal{D}$, with complete input trace $\overline{z}; \overline{x}$. Let J' denote the feedback-reachable subdiagram of K. Then $J \subseteq J'$, because every element of J is the result of a computation sequence in J, and $J \subseteq K$ means that every computation sequence in J is a feedback computation sequence in K. Also, $J' \subseteq J$, because each element of a feedback computation sequence in K is consistent with J, and has input trace a prefix of $\overline{x}$, hence is in J because J is a behavior of $\{D\}_{\doteq Z}$. Thus $J = J'$.

We claim that J is cofinal in K. To establish this, we show by induction on n, that if t is a element of K, and t is the result t_n of a computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$ for K, then there exists an element u_n of J with $t_n \preceq u_n$.

In the basis case, we have $t_0 = \perp$, so we may take $u_0 = \perp$.

For the induction step, suppose we have established the result for n, and consider the case of $n+1$. Then t is the result t_{n+1} of a computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_{n+1}$ of K. Applying the induction hypothesis to the computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$, we obtain an element u_n of J with $t_n \preceq u_n$. Let $v = t_{n+1} \uparrow t_n$, then v is a transition of D by definition of a computation sequence. (See Figure 5.) Now, $\lambda_Z^{\mathrm{in}}(t_{n+1}) \preceq \overline{z}$, because $t_{n+1} \in K$ and $\overline{\lambda}_Z^{\mathrm{in}}(K) = \overline{z}$. Since we also have $\overline{\lambda}_Z^{\mathrm{in}}(J) = \overline{z}$, and u_n is an element of J, there must exist an element u'_n of J, with $u_n \preceq u'_n$ and $\lambda_Z^{\mathrm{in}}(t_{n+1}) \preceq \lambda_Z^{\mathrm{in}}(u'_n)$. Let $v' = v \uparrow (u'_n \uparrow t_n)$; then $\lambda_Z^{\mathrm{in}}(v') = \epsilon$. Moreover, v' is a transition of D because v is. Let v'' be the pure-input transition of D with $\mathrm{dom}(v'') = \mathrm{dom}(v')$ and $\lambda^{\mathrm{in}}(v'') = \lambda_Z^{\mathrm{out}}(v'); \epsilon_X$. Let $w = v' \vee v''$, which exists by Theorem 4.1. Let $u_{n+1} = u'_n w$, then u_{n+1} is feedback-reachable, and $t_{n+1} \preceq u_{n+1}$.

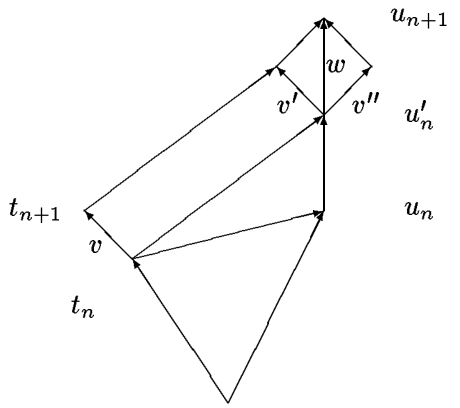

Figure 5: Proof of Theorem 4.4

To complete the induction step, it remains to be shown that $u_{n+1} \in J$. It suffices, since J is a behavior of $\{D\}_{\dot{=}Z}$, to show that $\lambda_X(u_{n+1}) \preceq \overline{x}$ and that u_{n+1} is consistent with J. That u_{n+1} is consistent with J is clear, since $u_{n+1} = t_{n+1} \vee u'_n v''$, and both t_{n+1} and $u'_n v''$ are consistent with J. Also, $\lambda_X(u_{n+1}) \preceq \overline{x}$ holds, since $\lambda_X(u_{n+1}) = \lambda_X(t_{n+1} \vee u'_n v'') = \lambda_X(t_{n+1}) \vee \lambda_X(u'_n)$, and both t_{n+1} and u'_n are in K.

($\Leftarrow$) Suppose K is a behavior of $\mathcal{D}$. Let J be the set of feedback-reachable elements of K, and suppose J is cofinal in K. Then J and K have the same complete input trace, say $\overline{z}; \overline{x}$, with respect to λ. We claim that J is a behavior of $\{D\}_{\dot{=}Z}$; that is, J is maximal among all computations J' of $\{D\}_{\dot{=}Z}$ with $\overline{\lambda}_X(J') = \overline{x}$. To show this, we show that if t is a feedback-reachable element of D, consistent with J, and such that $\lambda_X(t) \preceq \overline{x}$, then $t \in K$. It then follows that $t \in J$ because J is the feedback-reachable subdiagram of K.

We proceed by induction on the length n of a feedback computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$ for D, with result t.

In the basis case, we have $t = t_0 = \perp$, hence $t \in K$.

For the induction step, suppose we have shown the result for n, and consider the case of $n+1$. Then t is the result t_{n+1} of a feedback computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_{n+1}$, and t is consistent with J. By the induction hypothesis, $t_n \in K$. Let $u = t_{n+1} \uparrow t_n$, then u is a transition of D and we have $\lambda^{\text{in}}_Z(t_{n+1}) = \lambda^{\text{out}}_Z(t_{n+1})$, by definition of a feedback computation sequence.

Now, u has a decomposition $u = v \vee w$, where v is the pure-input transition of D with $\text{dom}(v) = \text{dom}(u)$ and $\lambda^{\text{in}}(v) = \lambda^{\text{in}}_Z(u); \epsilon_X$. Then $t_n w$ is an element of D that is consistent with J, and has the property $\lambda^{\text{in}}_Z(t_n w) = \lambda^{\text{in}}_Z(t_n) \preceq \overline{z}$. Since J is cofinal in K and $t_n w$ is consistent with J, we must have $t_n w$ consistent with K, and hence in K, because K is a behavior. Thus, $\lambda^{\text{in}}_Z(t_{n+1}) = \lambda^{\text{out}}_Z(t_{n+1}) = \lambda^{\text{out}}_Z(t_n u) = \lambda^{\text{out}}_Z(t_n w) \preceq \overline{z}$. But then t_{n+1} must be in K, since it is consistent with J, hence with K, we have $\lambda^{\text{in}}_Z(t_{n+1}) \preceq \overline{z}$, and K is a behavior. ∎

4.4 Histories

An (X, Y)-*history* is a nonempty, join-closed, and directed subset H of $X \times Y$, with the following additional property: for each $x; y \in H$, there exists a sequence

$$\epsilon_X; \epsilon_Y = x_0; y_0 \preceq x_1; y_1 \preceq \ldots \preceq x_n; y_n = x; y,$$

such that for each k with $0 \leq k < n$, the trace $x_n; y_{n+1}$ is in H. We call such a sequence a *computation sequence* for H, and the trace $x; y$ its *result.* If H is an (X,Y)-history, then the ideal $\overline{x}; \overline{y} = \bigvee H \in \overline{X} \times \overline{Y}$ is called the *complete trace* of H, with $\overline{x}$ called the *complete input trace* and $\overline{y}$ called the *complete output trace.*

If J is a behavior of an (X,Y)-input/output diagram, then it is easy to see from the properties of such diagrams that the history $\lambda(J)$ of J is, in fact, an (X,Y)-history.

Suppose H is a $(Z \times X, Y \times Z)$-history. A *feedback computation sequence* for H is a computation sequence

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n,$$

for H. If $z; x; y; z$ is the result $z_n; x_n; y_n; z_n$ of a feedback computation sequence for H, then we say that $z; x; y; z$ is *feedback-reachable* in H. It is easy to see that the set of all $x; y; z \in X \times Y \times Z$ such that $z; x; y; z$ is feedback-reachable in H is an $(X, Y \times Z)$-history, and we denote it by $\{H\}_{\circlearrowleft Z}$.

Lemma 4.4.1 *Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times X, Y \times Z)$-input/output diagram. If H is the history of a behavior K of $\mathcal{D}$, then $\{H\}_{\circlearrowleft Z}$ is the history of the set of feedback-reachable elements of K.*

Proof – Let J be the set of feedback-reachable elements of K, and let $G = \{H\}_{\circlearrowleft Z}$.

To see that the history of J is a subset of G, suppose $x; y; z$ is in the history of J. Then $z; x; y; z$ is the trace of the result t_n of a feedback computation sequence $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$ for K. Let $z_k; x_k; y_k; z_k$ be the trace of t_k, for each k. We claim that the sequence

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n$$

is a feedback computation sequence for H, thus showing that $x; y; z = x_n; y_n; z_n \in \{H\}_{\circlearrowleft Z}$. We show this as follows: Let $u_k = t_{k+1} \uparrow t_k$; then u_k is a transition of D. By the properties of input/output diagrams, we may write $u_k = v_k \vee w_k$, where v_k is the pure-input transition of D with $\mathrm{dom}(v_k) = \mathrm{dom}(u_k)$ and trace $\lambda^{\mathrm{in}}(u_k); \epsilon_{Y \times Z}$. Then w_k has trace $\epsilon_{Z \times X}; \lambda^{\mathrm{out}}(u_k)$, so $t_k w_k$ has trace $z_k; x_k; y_{k+1}; z_{k+1}$. Since $t_k w_k \preceq t_{k+1} \in K$, we must have $t_k w_k \in K$, and hence $z_k; x_k; y_{k+1}; z_{k+1} \in H$. But this is exactly what is required to show that the $z_k; x_k; y_k; z_k$ form a feedback computation sequence for H.

Conversely, if $x; y; z \in G$, then $z; x; y; z$ is the result $z_n; x_n; y_n; z_n$ of a feedback computation sequence

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n$$

for H. We claim that there exists a feedback computation sequence

$$t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_n$$

for K, and nonnegative integers $0 = m_0, m_1, \ldots, m_n = n$, such that $z_k; x_k; y_k; z_k$ is the trace of t_{m_k}, for each k.

The construction proceeds by induction on k. For the basis case ($k = 0$), we take $m_0 = 0$, and $t_0 = \perp$. Suppose now, for some k with $0 \leq k < n$, that we have constructed m_k and $t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_{m_k} \in K$, such that t_{m_k} has trace $z_k; x_k; y_k; z_k$. By definition of a feedback sequence, we know that $z_k; x_k; y_{k+1}; z_{k+1}$ is in H, hence is the trace of an element u_k of K. Without loss of generality, we may assume that $t_k \preceq u_k$. Thus, we may obtain $v_0 \prec_1 v_1 \prec_1 \ldots \prec_1 v_p \in K$, with $v_0 = t_{m_k}$ and $v_p = u_k$. For $0 \leq i \leq p$, let w_i be the pure-input element of D with trace $\lambda_Z^{\mathrm{out}}(v_i); \epsilon_X; \epsilon_Y; \epsilon_Z$. Note that then $v_i \vee w_i$ exists for each i with $0 \leq i \leq p$, and we have $v_i \vee w_i \prec_1 v_{i+1} \vee w_{i+1}$ for each i

with $0 \leq i < p$, by the properties of input/output diagrams. Moreover, $v_i \vee w_i \in K$ for each i with $0 \leq i \leq p$ because $v_i \in K$ and $\lambda^{\mathrm{in}}(w_i) \preceq z_{k+1}$, which is a prefix of the complete input trace of H, hence of K. Let $m_{k+1} = m_k + p$, and let $t_{m_k+i} = v_i \vee w_i$ for each i with $0 < i \leq p$. Then

$$t_0 \prec_1 t_1 \prec_1 \ldots \prec_1 t_{m_k} \prec_1 t_{m_k+1} \prec_1 \ldots \prec_1 t_{m_{k+1}}$$

is the required feedback computation sequence for K, thus completing the induction step and the proof. ∎

Theorem 4.5 *Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times X, Y \times Z)$-input/output diagram. Then an $(X, Y \times Z)$-history G, with complete trace $\overline{x}; \overline{y}; \overline{z}$, is a history of $\{\mathcal{D}\}_{\doteq Z}$ iff $G = \{H\}_{\circlearrowleft Z}$ for some history H of $\mathcal{D}$ with complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$.*

Proof – If G is a history of $\{\mathcal{D}\}_{\doteq Z}$, with complete trace $\overline{x}; \overline{y}; \overline{z}$, then G is the history of some behavior J of $\{\mathcal{D}\}_{\doteq Z}$. By Theorem 4.4, there exists a behavior K of $\mathcal{D}$ such that J is the set of feedback-reachable elements of K, and J is cofinal in K. Let H be the history of K; then H has complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$ by Lemma 4.3.2. Moreover, $G = \{H\}_{\circlearrowleft Z}$ by Lemma 4.4.1.

Conversely, suppose $G = \{H\}_{\circlearrowleft Z}$ for some history H of $\mathcal{D}$. Suppose G has complete trace $\overline{x}; \overline{y}; \overline{z}$ and H has complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$. Then H is the history of some behavior K of $\mathcal{D}$. By Lemma 4.4.1, G is the history of the set J of feedback-compatible elements of K, from which it follows by Lemma 4.3.2 that J is cofinal in K. Thus, J is a behavior of $\{\mathcal{D}\}_{\doteq Z}$, and G is a history of $\{\mathcal{D}\}_{\doteq Z}$.
∎

4.5 Scenarios

An (X, Y)-*Kahn function* is a continuous function ϕ from an initial segment (a nonempty, downward-closed, and directed-complete subset) of $\mathrm{dom}(\phi)$ of $\overline{X}$ to $\overline{Y}$. If $\mathrm{dom}(\phi) = \overline{X}$, then ϕ is called *total.* If U is an initial segment of $\overline{X}$, then the set of all (X, Y)-Kahn functions with domain U forms a directed-complete poset under the argumentwise ordering. We use the traditional notations $\sqsubseteq$ and $\sqcup$ to denote this ordering and the associated supremum operation, respectively.

Suppose ϕ is a $(Z \times X, Y \times Z)$-Kahn function. We say that ϕ is *feedback-compatible* if $\overline{y}; \overline{z} \in \phi(\mathrm{dom}(\phi))$ implies $\overline{z}; \overline{x} \in \mathrm{dom}(\phi)$ for all $\overline{x} \in \mathrm{dom}(\phi)$. Note that total $(Z \times X, Y \times Z)$-Kahn functions are always feedback-compatible.

If ϕ is feedback compatible, then define the *feedback functional* Φ *associated with* ϕ, to be the functional

$$\Phi : [\pi_{\overline{X}}(\mathrm{dom}(\phi)) \to \overline{Y} \times \overline{Z}] \to [\pi_{\overline{X}}(\mathrm{dom}(\phi)) \to \overline{Y} \times \overline{Z}]$$

that takes each $(X, Y \times Z)$-Kahn function $\psi : \pi_{\overline{X}}(\mathrm{dom}(\phi)) \to \overline{Y} \times \overline{Z}$ to an $(X, Y \times Z)$-Kahn function

$$\Phi(\psi) = \phi \circ ((\pi_{\overline{Z}} \circ \psi) \times \mathrm{id}_{\mathrm{dom}(\psi)}).$$

The feedback-compatibility of ϕ guarantees that Φ is well-defined, and it is easily verified that Φ is continuous. Define $\{\phi\}_{\circlearrowleft Z}$ to be the least fixed point of Φ.

Lemma 4.5.1 *If ϕ is a feedback-compatible $(Z \times X, Y \times Z)$-Kahn function, and Φ is the associated feedback functional, then*

$$\{\phi\}_{\circlearrowleft Z} = \bigsqcup_{k=0}^{\infty} \phi^{(k)},$$

where $\phi^{(0)}$ is the identically ϵ function, and $\phi^{(k+1)} = \Phi(\phi^{(k)})$ for each $k \geq 0$.

Proof – Standard. ∎

An (X, Y)-*scenario* is an (X, Y)-Kahn function whose domain is directed. The pair

$$\left(\bigvee \mathrm{dom}(\phi)\right);\left(\bigvee \phi(\mathrm{dom}(\phi))\right)$$

is called the *complete trace* of ϕ, with $\bigvee \mathrm{dom}(\phi)$ called the *complete input trace* and $\bigvee \phi(\mathrm{dom}(\phi))$ called the *complete output trace.*

Suppose H is an (X, Y)-history, with complete trace $\overline{x}_0; \overline{y}_0$. Then H determines an (X, Y)-scenario

$$\phi : \{\overline{x} \in \overline{X} : \overline{x} \preceq \overline{x}_0\} \rightarrow \overline{Y},$$

according to the definition

$$\phi(\overline{x}) = \bigvee\{y \in Y : \exists x; y \in H, x \preceq \overline{x}\}.$$

Intuitively, a scenario represents some information about how inputs precede outputs in a single computation. That is, $\phi(\overline{x}) = \overline{y}$ iff, in a single computation with scenario ϕ, the input trace $\overline{x}$ "enables" the output trace $\overline{y}$ in the sense that it is possible for arbitrarily large finite prefixes y of $\overline{y}$ to be generated in response to inputs that are finite prefixes of $\overline{x}$.

Brock and Ackerman [7, 6] have defined "scenario" based on the notion of when finite inputs "must precede" finite outputs. The two notions of scenario are evidently equivalent, since in a computation with complete trace $\overline{x}_0; \overline{y}_0$, $\overline{x}$ "enables" $\overline{y}$ iff every finite prefix x of $\overline{x}_0$ that "must precede" some finite prefix y of $\overline{y}$ is already a prefix of $\overline{x}$, and x "must precede" y iff for all $\overline{x} \preceq \overline{x}_0$, if $\overline{x}$ "enables" y, then $x \preceq \overline{x}$. We find that a definition of scenario based on "enables," rather than "must precede," is easier to relate to Kahn's continuous function model of processes.

Lemma 4.5.2 *Suppose H is a $(Z \times X, Y \times Z)$-history, with scenario ϕ. Let $G = \{H\}_{\circlearrowleft Z}$, and let ψ be the scenario of G. If ϕ is feedback-compatible, then $\psi = \{\phi\}_{\circlearrowleft Z}$.*

Proof – We show $(\Rightarrow)$ $\psi(\overline{x}) \preceq \{\phi\}_{\circlearrowleft Z}(\overline{x})$ for all $\overline{x} \in \overline{X}$, and $(\Leftarrow)$ ψ is a fixed point of the feedback functional associated with ϕ.

$(\Leftarrow)$ Suppose $\psi(\overline{x}) = \overline{y}; \overline{z}$. Let $y; z$ be an arbitrary finite prefix of $\overline{y}; \overline{z}$. Then there exists a feedback computation sequence

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n$$

for H, such that $x_n \preceq \overline{x}$ and $y; z \preceq y_n; z_n$. A simple induction shows that for each k with $0 \leq k < n$ we have

$$y_{k+1}; z_{k+1} \preceq \phi^{(k+1)}(\overline{x}),$$

where $\phi^{(k)}$ is as defined in Lemma 4.5.1. It follows that $y; z \preceq \{\phi\}_{\circlearrowleft Z}(\overline{x})$. Since $y; z$ was an arbitrary finite prefix of $\overline{y}; \overline{z}$, it follows that $\overline{y}; \overline{z} \preceq \{\phi\}_{\circlearrowleft Z}(\overline{x})$, as was to be shown.

$(\Rightarrow)$ Since we already know that $\psi \sqsubseteq \{\phi\}_{\circlearrowleft Z}$, to show that ψ is a fixed point of the feedback functional Φ associated with ϕ, it remains only to show that $\Phi(\psi) \sqsubseteq \psi$. That is, we must show that $\phi((\pi_{\overline{Z}}(\psi(\overline{x})); \overline{x}) \preceq \psi(\overline{x})$ for all $\overline{x} \in \overline{X}$. To show this, it suffices to show that for all $x; y; z \in G$, if $y'; z'$ is a finite prefix of $\phi(z; x)$, then there exists $x; y''; z'' \in G$, such that $y' \preceq y''$ and $z' \preceq z''$.

Now, if $x; y; z \in G$, then $x; y; z$ is the result $x_n; y_n; z_n$ of a feedback computation sequence

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n$$

for H. If $y'; z' \preceq \phi(z; x)$, then $z; x; y''; z'' \in H$ for some $y'' \in Y$ and $z'' \in Z$ with $y' \preceq y''$ and $z' \preceq z''$. This shows that

$$z_0; x_0; y_0; z_0 \preceq z_1; x_1; y_1; z_1 \preceq \ldots \preceq z_n; x_n; y_n; z_n \preceq z''; x; y''; z''$$

is a feedback sequence for H. It follows that $x; y''; z''$ is in G. ▮

Theorem 4.6 *Suppose $\mathcal{D} = (D, \lambda)$ is a $(Z \times X, Y \times Z)$-input/output diagram. Then an $(X, Y \times Z)$-scenario ψ, with complete trace $\overline{x}; \overline{y}; \overline{z}$, is a scenario of $\{\mathcal{D}\}_{\doteq Z}$ iff $\psi = \{\phi\}_{\circlearrowleft Z}$, where ϕ is a scenario of $\mathcal{D}$ with complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$.*

Proof – If ψ is a scenario of $\{\mathcal{D}\}_{\doteq Z}$, with complete trace $\overline{x}; \overline{y}; \overline{z}$, then it is the scenario of some history G of $\{\mathcal{D}\}_{\doteq Z}$, with the same complete trace. By Theorem 4.5 there exists a history H of $\mathcal{D}$, with complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$, such that $G = \{H\}_{\circlearrowleft Z}$. If ϕ is the scenario of H, then ϕ is obviously feedback-compatible, and $\psi = \{\phi\}_{\circlearrowleft Z}$ by Lemma 4.5.2.

Conversely, if $\psi = \{\phi\}_{\circlearrowleft Z}$ has complete trace $\overline{x}; \overline{y}; \overline{z}$, where ϕ is a scenario of $\mathcal{D}$ with complete trace $\overline{z}; \overline{x}; \overline{y}; \overline{z}$, then ϕ is the scenario of some history H of $\mathcal{D}$, and is clearly feedback-compatible. By Theorem 4.5, $\{H\}_{\circlearrowleft Z}$ is a history of $\{\mathcal{D}\}_{\doteq Z}$, and by Lemma 4.5.2, $\{H\}_{\circlearrowleft Z}$ has scenario ψ. ▮

4.6 Input/Output Relations

Given an (X, Y)-input/output diagram $\mathcal{D}$, define the *input/output relation* $\mathrm{Reln}(\mathcal{D})$ of $\mathcal{D}$ to be the set of all complete traces of scenarios of $\mathcal{D}$.

Lemma 4.6.1 *Suppose $\mathcal{D}$ is an (X, Y)-input/output diagram. Then* $\mathrm{Reln}(\mathcal{D})$ *is:*

- *(Total) For all $\overline{x} \in \overline{X}$, there exists $\overline{y} \in \overline{Y}$ such that $\overline{x}; \overline{y} \in \mathrm{Reln}(\mathcal{D})$.*
- *(Monotone) If $\overline{x}; \overline{y} \in \mathrm{Reln}(\mathcal{D})$, and $\overline{x} \preceq \overline{x}'$, then there exists $\overline{y}'$, with $\overline{y} \preceq \overline{y}'$, such that $\overline{x}'; \overline{y}' \in \mathrm{Reln}(\mathcal{D})$.*

Moreover, if $\mathcal{D}$ is a Kahn diagram, then $\mathrm{Reln}(\mathcal{D})$ *is (the graph of) a total (X, Y)-Kahn function, and the scenarios of $\mathcal{D}$ are exactly the restrictions of* $\mathrm{Reln}(\mathcal{D})$ *to directed initial segments of $\overline{X}$.*

Proof – Straightforward from Lemma 4.3.1. ▮

If $R \subseteq \overline{X} \times \overline{Y}$, then let R_{fin} denote the set of finite prefixes of elements of R. The relation R is called *continuous* if, for all $\overline{x} \in \overline{X}$, whenever U is a maximal directed subset of

$$\{y \in Y : \exists x \preceq \overline{x}, x; y \in R_{\mathrm{fin}}\},$$

then $\overline{x}; (\bigvee U) \in R$. Note that if R is the graph of a total (X, Y)-Kahn function, then R is continuous.

Lemma 4.6.2 *Suppose $R \subseteq \overline{X} \times \overline{Y}$ is total, monotone, and continuous. Then there exists an (X, Y)-machine $\mathcal{M}$ with R as its input/output relation.*

Proof – Define the machine $\mathcal{M}$ as follows:

- Proper states: all elements $x;y \in R_{\mathrm{fin}}$. Take $\epsilon_X;\epsilon_Y$ as the start state.
- Proper transitions: all pairs $(x;y,v) \in R_{\mathrm{fin}} \times Y$, such that $x;yv \in R_{\mathrm{fin}}$. Define $\mathrm{dom}(x;y,v) = x;y$ and $\mathrm{cod}(x;y,v) = x;yv$. Take the transitions $(x;y,\epsilon_Y)$ as identities.
- Residual: define $(x;y,v)$ and $(x;y,v')$ to be consistent iff v and v' are consistent, in which case define
$$(x;y,v) \uparrow (x;y,v') = (x;yv', v \uparrow v').$$
- Output map: define $\lambda(x;y,v) = v$.
- Input map: define $\delta_{x'}(x;y,v) = (xx';y,v)$.

It is not difficult to verify that $\mathcal{M}$ is an (X,Y)-machine. Moreover, a set $H \subseteq X \times Y$ is the history of a behavior of M exactly when $\pi_X(H)$ is an ideal $\overline{x}$ of X, and $\pi_Y(H)$ is a maximal directed subset of
$$\{y \in Y : \exists x \preceq \overline{x}, x;y \in R_{\mathrm{fin}}\}.$$
Since R is continuous, it follows that M has R as its input/output relation. ∎

Lemma 4.6.3 *Suppose ϕ is a total $(Z \times X, Y \times Z)$-Kahn function with complete trace $\overline{z};\overline{x};\overline{y};\overline{z}$, ψ is the restriction of ϕ to prefixes of $\overline{z};\overline{x}$, and $\{\psi\}_{\circlearrowleft Z}$ has trace $\overline{x};\overline{y};\overline{z}$. Then $\{\psi\}_{\circlearrowleft Z}$ is the restriction of $\{\phi\}_{\circlearrowleft Z}$ to prefixes of $\overline{x}$.*

Proof – A simple induction shows that for each $k \geq 0$ the scenario $\psi^{(k)}$ is the restriction to prefixes of $\overline{x}$ of $\phi^{(k)}$, where $\psi^{(k)}$ and $\phi^{(k)}$ are as defined in Lemma 4.5.1. From this, the result follows by continuity of restriction. ∎

Lemma 4.6.4 *Suppose ϕ is a total $(Z \times X, Y \times Z)$-Kahn function. Then $\{\phi\}_{\circlearrowleft Z}(\overline{x}) = \overline{y};\overline{z}$ iff the restriction ψ of ϕ to prefixes of $\overline{z};\overline{x}$ is feedback-compatible, and $\{\psi\}_{\circlearrowleft Z}$ has complete trace $\overline{x};\overline{y};\overline{z}$.*

Proof – Suppose $\{\phi\}_{\circlearrowleft Z}(\overline{x}) = \overline{y};\overline{z}$. Let ψ be the restriction of ϕ to prefixes of $\overline{z};\overline{x}$. Then $\phi(\overline{z};\overline{x}) = \overline{y};\overline{z}$, so ψ is feedback-compatible. By Lemma 4.6.3, $\{\psi\}_{\circlearrowleft Z}$ is the restriction of $\{\phi\}_{\circlearrowleft Z}$ to prefixes of $\overline{x}$, hence $\overline{x};\overline{y};\overline{z}$ is the complete trace of $\{\psi\}_{\circlearrowleft Z}$.

Conversely, suppose the restriction ψ of ϕ to prefixes of $\overline{z};\overline{x}$ is feedback-compatible, and $\{\psi\}_{\circlearrowleft Z}$ has trace $\overline{x};\overline{y};\overline{z}$. By Lemma 4.6.3, $\{\psi\}_{\circlearrowleft Z}$ is the restriction to prefixes of $\overline{x}$ of $\{\phi\}_{\circlearrowleft Z}$. Since $\{\psi\}_{\circlearrowleft Z}$ has complete trace $\overline{x};\overline{y};\overline{z}$, it follows that $(\{\psi\}_{\circlearrowleft Z})(\overline{x}) = \overline{y};\overline{z}$, and hence $\{\phi\}_{\circlearrowleft Z}(\overline{x}) = \overline{y};\overline{z}$. ∎

Theorem 4.7 *(Kahn Principle) Suppose $\mathcal{D}$ is a $(Z \times X, Y \times Z)$-Kahn diagram. Then*
$$\mathrm{Reln}(\{\mathcal{D}\}_{\doteq Z}) = \{\mathrm{Reln}(\mathcal{D})\}_{\circlearrowleft Z}.$$

Proof – By definition of Reln, $\overline{x};\overline{y};\overline{z} \in \mathrm{Reln}(\{\mathcal{D}\}_{\doteq Z})$ iff there exists a scenario of $\{\mathcal{D}\}_{\doteq Z}$ with complete trace $\overline{x};\overline{y};\overline{z}$. By Lemma 4.6.4, this is true iff there exists a feedback-compatible scenario ϕ of $\mathcal{D}$, with trace $\overline{z};\overline{x};\overline{y};\overline{z}$, such that $\{\phi\}_{\circlearrowleft Z}$ has trace $\overline{x};\overline{y};\overline{z}$. By Lemma 4.6.3, this is true iff the restriction ϕ of $\mathrm{Reln}(\mathcal{D})$ to prefixes of $\overline{z};\overline{x}$ is feedback-compatible, has trace $\overline{z};\overline{x};\overline{y};\overline{z}$, and $\{\phi\}_{\circlearrowleft Z}$ has trace $\overline{x};\overline{y};\overline{z}$. By Lemma 4.6.4, this is true iff $\overline{x};\overline{y};\overline{z} \in \{\mathrm{Reln}(\mathcal{D})\}_{\circlearrowleft Z}$. ∎

5 Discussion

The author was led to define concurrent transition systems because of the apparent difficulty of establishing relationships between operational and denotational models of concurrent computation. The problems, of finding a natural characterization of a large class of dataflow-like processes with functional behavior, and of proving that the feedback operation on such processes satisfies the Kahn Principle, served as primary motivating examples. Before trying the concurrent transition system approach reported here, an attempt was made to try to solve these problems using a model of processes based on ordinary (nondeterministic) labeled transition systems. Although other researchers have shown how various parallel composition operations of CCS and CSP can be given reasonably natural definitions in such a model, our situation is somewhat different, because we have drawn a distinction between input and output, and because we are interested in infinite computations as well as finite ones.

There were two difficulties that seemed inherent in an approach based on ordinary transition systems. The first difficulty arose from the fact that, although we are interested in infinite computations of a system, we are only interested in those infinite computations that are "completed" in the sense that each process produces all the output implied by the input it has received. The usual method of handling this is to distinguish between "fair" and "unfair" computations. This approach leads to technical problems, as pointed out in Section 1. The second difficulty with the nondeterministic transition system approach was that the notion of "primitive" or "atomic" steps of a process seemed not to behave smoothly with respect to the feedback operation. One can see the problem by considering an "identity" process, which simply passes its input through unchanged to its output. One would like to have the atomic steps of this process correspond to the receipt of input and the issuance of output. Now, consider what happens when the output of the identity process is fed back to its input. The "intuitively correct" result of this construction is a process that produces no output. It seems most natural to define the atomic steps of the fed-back identity to correspond to the simultaneous issuance of output and the absorption of that output as feedback input. However, the question arises of how to define the construction in such a way that "nonintuitive" computations, in which output is produced, are avoided.

In retrospect, concurrent transition systems seem to provide exactly the right structure to circumvent the difficulties mentioned above. The fairness problem is solved, in the concurrent transition system approach, by replacing the notion "fair computation sequence" by the more convenient notion "behavior" or "maximal ideal." The atomic step problem is solved by restricting attention to a class of processes whose transitions have pure-input/output decompositions. In essence, the existence of such decompositions means that there is an inherent delay of one atomic step between input and output, and this allows nonintuitive computations to be avoided. The existence of pure-input/output decompositions is easily and naturally expressed with concurrent transition systems, whereas it is not clear how the same could be done with ordinary transition systems.

5.1 Related Work

As mentioned in Section 2, the defining axioms for concurrent transition systems are satisfied by the derivation relation of the λ-calculus, and the computation category construction is an abstract version of a construction that has already been found useful in that setting. The goal of the λ-calculus work [22, 5], and the extension of this work to term-rewriting systems [15], is to try to find reduction strategies that are optimal in the sense that only redexes that are "needed" are contracted, and each needed redex is contracted only once. The main theorem one tries to prove is that every derivation is in a sense equivalent to an optimal derivation. To make the notions "needed redex" and "equivalent derivations" precise, the "residual" operation is defined. Intuitively, the residual operation serves to keep track of what happens to one redex when others are contracted. A redex is "needed" if it (or its residuals) must be contracted in any derivation sequence that leads to a normal form. Two derivation sequences are regarded as equivalent iff the same set of reductions is performed in each, where the notion "same set of reductions" is interpreted modulo residuals. The residual operation for CTS's was introduced for an essentially similar purpose: to keep track of what happens to a particular atomic transition (say for one process) of a system, when other atomic steps (say for other, concurrently executing processes) are executed. Two computation sequences are regarded as equivalent representatives of the same concurrent computation if they contain the "same set of atomic steps."

A difference between the CTS and and term-rewriting settings are that in the former we regard inconsistent pairs of coinitial transitions as meaningful, whereas in the latter one is usually interested only in confluent or Church-Rosser systems. Also, with CTS's we are interested in nonterminating computations, whereas in the rewriting situation one is primarily interested in terminating or normalizing computations.

Several authors have investigated algebraic structures for modeling concurrency that seem related to concurrent transition systems. Winskel [42] defines the notion of a "synchronization tree," which is a (possibly infinite) tree whose arcs are labeled with elements of a "synchronization algebra." In [41], labeled *event structures* [29] are used in place of labeled trees. Using various synchronization algebras, Winskel is able to show several notions of parallel composition from CCS and CSP to be special cases of a single definition. It is clear that Winskel's trees are special cases of our computation diagrams. Also, Winskel's synchronization algebras are rather similar to our trace algebras. Specifically, a trace algebra can be regarded as a synchronization algebra if we identify Winskel's $*$ with our ϵ, and Winskel's operation $\bullet$ with our operation $\vee$. It is not possible, in general, to regard a synchronization algebra as a trace algebra, since the latter are somewhat more highly structured. We use trace algebras to label the transitions of CTS's in essentially the same way as Winskel uses synchronization algebras to label trees. However, we find it an advantage that trace algebras are a particular kind of CTS. By regarding Winskel's synchronization trees as special cases of our synchronization diagrams, essentially the same parallel composition constructions can be carried out in our framework.

Event structures and CTS's can be related as follows: Given a CTS with start state (C, ι), it is straightforward to make the set of elements of $\mathrm{Diag}(C, \iota)$ into an event structure by defining the "consistent" sets of elements to be the finite sets that are consistent in the sense we have defined here, and defining a consistent set T to "enable" an element t iff there is a subset U of T such that $(\bigvee U) \prec_1 t$. Conversely, the set of "configurations" of an event structure is a partially ordered set in which every finite subset with an upper bound has a least upper bound, and hence is easily made into a complete CTS, by taking configurations as proper states and the ordering relation as the set of proper transitions. In a sense, CTS's can be thought of as a somewhat more primitive operational model than event structures, since in the former one is free to designate the set of states, whereas in the latter, states are always obtained as configurations.

Main and Benson [25] use ideas from multilinear algebra to model nondeterministic and concurrent processes without iteration or recursion. An important role is played by "positive semi-rings," whose formal properties are closely related the trace algebras used in the present paper. Essentially, a trace algebra Z is a positive semiring in which a left-cancellation law holds for multiplication, and in which there is a further connection between addition and multiplication; namely, addition is least upper bound with respect to the prefix order induced by multiplication.

Arbib and Manes [2] have developed a categorical theory of automata, which generalizes several classical situations. They generalize the notion of an "action" or "transition map" as a function $\delta : Q \times X \to Q$ to the notion of a "dynamics," which is a morphism $\delta : X(Q) \to Q$, where Q is an object of an arbitrary category $\mathbf{K}$, and X is an endofunctor of $\mathbf{K}$. Arbib and Manes' theory is applied to "port automata" in [37]. In that paper, concurrency is modeled by interleaving, and the issue of fair infinite computations is not considered. It would be nice if the definition of "action" we have given here could be shown to be a special case of Arbib and Manes' dynamics. However, we have yet to identify the proper endofunctor X of $\mathbf{CTS}$ to achieve this goal. The product-forming functor $(- \times X)$ does not yield a general enough class of dynamics.

The work of Winkowski [39, 40], is motivated by considerations in the theory of Petri nets. In [39], Winkowski defines the notion of a "behavior algebra," which is a category equipped with (among other things) a partial binary operation $+$ on the arrows of the category, representing independent concurrent composition. The properties of a behavior algebra are similar in many respects to those enjoyed by the computation categories defined in this paper. However, the theory of computation categories appears to be somewhat simpler than that of behavior algebras, primarily due to the fact that in computation categories there is a connection between concurrency and pushouts. The existence of this connection means that the concurrency information in a computation category can be obtained entirely from the structure of the category itself, without requiring the specification of additional information such as the operation $+$ of a behavior algebra. It also makes possible the definition of computations as ideals, which is substantially simpler than the definition of "histories" given by Winkowski.

Staples and Nguyen [34] define a dataflow-like model in which a process is represented by a partially ordered set whose elements are labeled by "histories" ("traces," in our terminology). Processes are required to satisfy a collection of axioms, which appear related to the properties enjoyed by synchronization diagrams in the present paper. It would seem that by taking an input/output synchronization diagram and equipping the cpo of its computations with the map that takes each computation to its complete trace, one obtains a structure that is similar to the processes of Staples and Nguyen, both in formal properties and in intuitive content. However, there is not an exact correspondence, since one of Staples and Nguyen's axioms concerns greatest lower bounds, whose existence we have not found it necessary to assume.

Labella and Pettorossi [21] have given categorical characterizations of various operations of CCS and CSP. In their approach they take as given a semantics of these languages defined in terms of equivalence classes of trees. A suitable definition of morphism makes the set of all these equivalence classes into a category, in which their characterizations are valid. The characterizations they obtain are not particularly simple, and one is not left with the feeling they are likely to translate to categories obtained from other concurrent programming languages. In contrast, in the present paper we hope that by defining a model in which simple categorical constructions appear to correspond to intuitively meaningful semantic operations on processes, we can use the same model to define the semantics of a number of different concurrent programming languages.

5.2 Directions for Future Research

One obvious avenue for future research is to extend the machine model to include a way of defining machines recursively. Presumably, a recursive definition of an (X, Y)-machine would denote a limit of an inverse system generated by a suitable continuous endofunctor on $\mathbf{Mach}_{X,Y}$. To properly develop this idea, we have to establish that such a limit construction would produce a machine with the intuitively correct set of computations. We also have to establish the continuity of a set of network-building operations, such as the parallel product, relabeling, and feedback operations defined in this paper.

Although the machine model defined here is capable of representing a large class of processes, including processes with functional input output behavior and an "unfair merge" process, it is possible to show that "fair merge" cannot be modeled. In addition, it is impossible to model processes that have "conflicts" or "race conditions" between input and output. An interesting question is whether it is possible to generalize our definitions in a natural way, so that a larger class of processes can be modeled. One way to approach this is to investigate classes of automata obtained by weakening some of the conditions of Theorem 4.1.

In Section 4, we pointed out that the feedback operation on machines, when mapped to automata and synchronization diagrams, could be characterized as right-adjoint to a relabeling functor. The parallel product of automata can also be characterized in a similar way. This phenomenon suggests the idea of defining a "process algebra" to be an (X, Y)-indexed collection of categories $\mathbf{Proc}_{X,Y}$, and to require that operations on processes be defined as adjoints to various naturally occurring functors. The advantages of such an approach include the ability to compare the concrete form taken by the "same operations" in different process algebras, and automatic proofs of continuity of operations arising from the adjoint characterizations. However, it is not clear whether such an approach is feasible, since we do not yet have an adjoint characterization of the feedback operation on machines, nor do we know whether it is possible to impose useful categorical structure on the behavior, history, and input/output relation models.

It would be nice to understand better the relationships between the CTS-based models defined in this paper and other models of concurrency, especially Petri Nets. One question here would be to see how much of the modeling power of Petri Nets is shared by CTS's, which are somewhat more abstract. Comparisons of input/output automata with labeled transition system models of CCS and CSP would also be useful. An interesting question is how the notion of "bisimulation" [30, 28], which is fundamental for ordinary transition systems, might be reasonably generalized to CTS's.

Finally, the full abstraction problem for machines remains open. Although we were not able to solve this problem in this paper, we have been able to make the problem more concrete by establishing the existence of a seemingly natural "fully abstract" algebra of processes. Moreover, we feel that the information about the feedback operation we have obtained is likely to be useful in ultimately resolving this important question.

Acknowledgement

I am grateful to the anonymous referees for their careful reading and perceptive comments.

References

[1] S. Abramsky. Experiments, powerdomains, and fully abstract models for applicative multiprogramming. In *Foundations of Computation Theory*, pages 1–13, Springer-Verlag. Volume 158

of *Lecture Notes in Computer Science*, 1983.

[2] M. A. Arbib and E. G. Manes. *Arrows, Structures, and Functors: The Categorical Imperative.* Academic Press, 1975.

[3] R. J. Back and N. Mannila. A refinement of Kahn's semantics to handle nondeterminism and communication. In *Proc. ACM Symposium on Principles of Distributed Computing*, pages 111–120, 1982.

[4] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, 1981.

[5] G. Berry and J.-J. Lévy. Minimal and optimal computations of recursive programs. *Journal of the ACM*, 26(1):148–175, January 1979.

[6] J. D. Brock. *A Formal Model of Non-Determinate Dataflow Computation.* PhD thesis, Massachusetts Institute of Technology, 1983. Available as MIT/LCS/TR-309.

[7] J. D. Brock and W. B. Ackerman. Scenarios: a model of non-determinate computation. In *Formalization of Programming Concepts*, pages 252–259, Springer-Verlag. Volume 107 of *Lecture Notes in Computer Science*, 1981.

[8] S. D. Brookes. On the relationship of ccs and csp. In *ICALP 83*, Springer Verlag, 1983.

[9] S. D. Brookes and W. C. Rounds. Behavioral equivalence relations induced by programming logics. In *Proceedings of ICALP 83*, 1983.

[10] A. A. Faustini. An operational semantics for pure dataflow. In *Automata, Languages, and Programming, 9th Colloquium*, pages 212–224, Springer-Verlag. Volume 140 of *Lecture Notes in Computer Science*, 1982.

[11] I. Guessarian. *Algebraic Semantics.* Volume 99 of *Lecture Notes in Computer Science*, Springer Verlag, 1981.

[12] M. Hennessy and R. de Nicola. Testing equivalences for processes. In *ICALP 1983*, Springer-Verlag. Volume 154 of *Lecture Notes in Computer Science*, 1983.

[13] H. Herrlich and G. E. Strecker. *Category Theory. Sigma Series in Pure Mathematics*, Heldermann Verlag, 1979.

[14] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–676, 1978.

[15] G. Huet. Formal structures for computation and deduction (first edition). May 1986. Unpublished manuscript. INRIA, France.

[16] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing 74*, North-Holland, 1974.

[17] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing 77*, North-Holland, 1977.

[18] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 337–366, North-Holland. 1978.

[19] R. M. Keller and P. Panangaden. Semantics of networks containing indeterminate operators. In *Seminar on Concurrency*, pages 479–496, Springer-Verlag. Volume 197 of *Lecture Notes in Computer Science*, 1984.

[20] J. N. Kok. Denotational semantics of nets with nondeterminism. In *ESOP 86*, pages 237–249, Springer-Verlag. Volume 213 of *Lecture Notes in Computer Science*, March 1986.

[21] A. Labella and A. Pettorossi. Categorical models for handshaking communications. In *Annales Societatis Mathematicae Polonae. SERIES IV: Fundamenta Informaticae*, 1985.

[22] J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda Calcul.* PhD thesis, Université Paris VII, 1978.

[23] S. Mac Lane. *Categories for the Working Mathematician.* Volume 5 of *Graduate Texts in Mathematics*, Springer Verlag, 1971.

[24] D. B. MacQueen. *Models for Distributed Computing.* Technical Report 351, INRIA, 1979.

[25] M. G. Main and D. B. Benson. Functional behavior of nondeterministic and concurrent programs. *Information and Control*, 62:144–189, 1984.

[26] A. Mazurkiewicz. Trace theory. In *Advanced Course on Petri Nets*, GMD, Bad Honnef, September 1986.

[27] R. Milner. *A Calculus of Communicating Systems.* Volume 92 of *Lecture Notes in Computer Science*, Springer Verlag, 1980.

[28] R. Milner. Lectures on a calculus for communicating systems. In *Seminar on Concurrency*, pages 197–220, Springer-Verlag. Volume 197 of *Lecture Notes in Computer Science*, 1984.

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

[30] D. M. R. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, Springer-Verlag. Volume 104 of *Lecture Notes in Computer Science*, 1981.

[31] D. M. R. Park. The "fairness problem" and nondeterministic computing networks. In *Proceedings, 4th Advanced Course on Theoretical Computer Science*, Mathematisch Centrum, 1982.

[32] V. R. Pratt. On the composition of processes. In *Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 213–223, January 1982.

[33] V. R. Pratt. The pomset model of parallel processes: unifying the temporal and the spatial. In *Seminar on Concurrency*, pages 180–196, Springer-Verlag. Volume 197 of *Lecture Notes in Computer Science*, July 1984.

[34] J. Staples and V. L. Nguyen. A fixpoint semantics for nondeterministic data flow. *Journal of the ACM*, 32(2):411–444, April 1985.

[35] E. W. Stark. *The Computation Category of a Concurrent Transition System.* Technical Report 86/08, State University of New York at Stony Brook Computer Science Dept., May 1986.

[36] E. W. Stark. Concurrent transition system semantics of process networks. In *Fourteenth ACM Symposium on Principles of Programming Languages*, pages 199–210, January 1987.

[37] M. Steenstrup, M. A. Arbib, and E. G. Manes. Port automata and the algebra of concurrent processes. *JCSS*, 27(1):29–50, 1983.

[38] P. S. Thiagarajan. Elementary net systems. In *Advanced Course on Petri Nets*, GMD, Bad Honnef, September 1986.

[39] J. Winkowski. An algebraic description of system behaviors. *Theoretical Computer Science*, 21:315–340, 1982.

[40] J. Winkowski. Behaviors of concurrent systems. *Theoretical Computer Science*, 12:39–60, 1980.

[41] G. Winskel. Event structures. In *Advanced Course on Petri Nets*, GMD, Bad Honnef, September 1986.

[42] G. Winskel. Synchronization trees. *Theoretical Computer Science*, 34:33–82, 1984.