

## CONCURRENT TRANSITION SYSTEMS

Eugene W. STARK\*

*State University of New York at Stony Brook, Stony Brook, NY 11794, U.S.A.*

Communicated by A. Meyer

Received March 1986

Revised February 1988

**Abstract.** *Concurrent transition systems* (CTS's), are ordinary nondeterministic transition systems that have been equipped with additional concurrency information. This concurrency information is specified in terms of a binary *residual operation* on transitions, which describes how certain pairs of transitions "commute." The defining axioms for a CTS generate a rich algebraic theory, which we develop in detail. 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 residual operation. The notion "computation tree" for ordinary transition systems generalizes to *computation diagram* for CTS's, leading to the convenient definition of *computations* of a CTS as the *ideals* of its computation diagram. A pleasant property of this definition 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 structural 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* [22] have been used as an operational semantics of concurrent processes. In typical formulations [9, 10], 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*,  $\Sigma$  is a set of *events*, not containing the distinguished symbol  $\epsilon$ , and

\* This research was supported in part by NSF Grant CCR-8702247.

$\Delta \subseteq Q \times (\Sigma \cup \varepsilon) \times Q$  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

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

where each  $q_k$  is in  $Q$ , each  $\sigma_k$  is in  $\Sigma \cup \{\varepsilon\}$ , and  $(q_k, \sigma_{k+1}, q_{k+1}) \in \Delta$  for each  $k$ . The string  $\sigma_1 \sigma_2 \dots \sigma_n$  is called the *trace* of the computation. Here we regard the set  $\Sigma$  as embedded in the free monoid  $\Sigma^*$  in the obvious way, and regard  $\varepsilon$  as the identity element of this monoid; thus  $\varepsilon$  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 [32] and CSP [17], 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” [37]—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 use the notion of “commuting paths” in a category to model the various interleaved views 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”  $G^*$  (the free category generated by  $G$ ), 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. For each state  $q$ , the comma category

$(q \downarrow G^*)$  is a poset category consisting of the finite computation sequences of  $G$  from state  $q$ , partially ordered by prefix. This is nothing more than the familiar “computation tree” of  $G$  from state  $q$ . The computations (both finite and infinite) of  $G$  can be identified with the ideals (nonempty, downward-closed, directed subsets) of the computation tree.

Now, the free categories generated by graphs are in a sense uninteresting, since there are no nontrivial commuting paths in such categories. Thus it would seem that we have obtained little more than an insignificant restatement of the definition of computation. 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. This concurrency information takes the form of a binary “residual” operation on transitions, whose purpose is, in essence, to specify which square diagrams of transitions “commute.” (The smooth development of the theory requires also the introduction of distinguished “identity” and “improper” transitions, which have special properties with respect to the residual operation.) Although not categories themselves, each CTS  $C$  freely generates a “complete CTS” or “computation category”  $C^*$ . The category  $C^*$  is constructed as a quotient of the free category generated by the underlying graph of  $C$ , modulo a congruence that relates two finite computation sequences exactly when they represent two interleaved views of the “same concurrent computation.” For each state  $q$ , the comma category  $(q \downarrow C^*)$  is a poset category, whose objects we interpret as the finite concurrent computations from state  $q$ . We call  $(q \downarrow C^*)$  the “computation diagram” of  $C$  from state  $q$ . Computations from state  $q$  are obtained as ideals of the computation diagram. It follows from the ideal construction that the set of all computations from initial state  $q$  is an algebraic directed-complete partial order. In certain circumstances, there is a coincidence between *fair* computation sequences and computations that are *maximal* under this 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 show that CTS has all small limits and coproducts, and is cartesian closed. We establish the existence of the computation categories and computation diagrams mentioned above, and prove some useful properties of computations.

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. Using our definition of computations as ideals, and regarding maximal ideals as “fair” or “completed” computations, 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 [2, 16, 28]. We also assume some familiarity with the theory of algebraic directed-complete partial orders, as used in denotational semantics. Reference [14] provides background on this topic. Preliminary versions of some of the results of this paper were reported in [41].

## 2. Concurrent transition systems

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

A *graph with identities* is a tuple  $G = (O, A, \text{dom}, \text{cod}, \text{id})$ , where  $O$  is a set of *objects*,  $A$  is a set of *arrows*,  $\text{dom}$ ,  $\text{cod}$  are functions from  $A$  to  $O$ , which map each arrow to its *domain* and *codomain*, respectively, and  $\text{id}: O \rightarrow A$  maps each  $q \in O$  to a distinguished *identity arrow*  $\text{id}_q$ . We require that  $\text{dom}(\text{id}_q) = \text{cod}(\text{id}_q) = q$  for all  $q \in O$ . Arrows  $t, u$  of  $G$  are called *composable* if  $\text{cod}(t) = \text{dom}(u)$  and *coinitial* if  $\text{dom}(t) = \text{dom}(u)$ . Let  $\text{Coin}(G)$  denote the set of all coinitial pairs of arrows of  $G$ .

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

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

- $G = (O, A, \text{dom}, \text{cod}, \text{id})$  is a graph with identities, called the *underlying graph*. The elements of  $O^\#$  (respectively  $O$ ) are called (respectively *proper*) *states* and the elements of  $A^\#$  (respectively  $A$ ) are called (respectively *proper*) *transitions*.
- $\uparrow: \text{Coin}(G^\#) \rightarrow A^\#$  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 coinitial  $t, u \in A^\#$ ,
  - (a)  $\text{dom}^\#(t \uparrow u) = \text{cod}^\#(u)$ ,
  - (b)  $\text{cod}^\#(t \uparrow u) = \text{cod}^\#(u \uparrow t)$ .
- (2) For all  $t: q \rightarrow r$  in  $A^\#$ ,
  - (a)  $\text{id}_q \uparrow t = \text{id}_r$ ,
  - (b)  $t \uparrow \text{id}_q = t$ ,
  - (c)  $t \uparrow t = \text{id}_r$ .
- (3) For all coinitial  $t, u, v \in A^\#$ ,  $(v \uparrow t) \uparrow (u \uparrow t) = (v \uparrow u) \uparrow (t \uparrow u)$ .
- (4) For all coinitial  $t, u \in A^\#$ , if  $t \uparrow u$  and  $u \uparrow t$  are both identities, then  $t = u$ .

Axiom (3) is called the *cube axiom*, and can be visualized as shown in Fig. 1. A *pre-CTS* is a structure  $(G, \uparrow)$  that satisfies all the CTS axioms except for axiom (4).

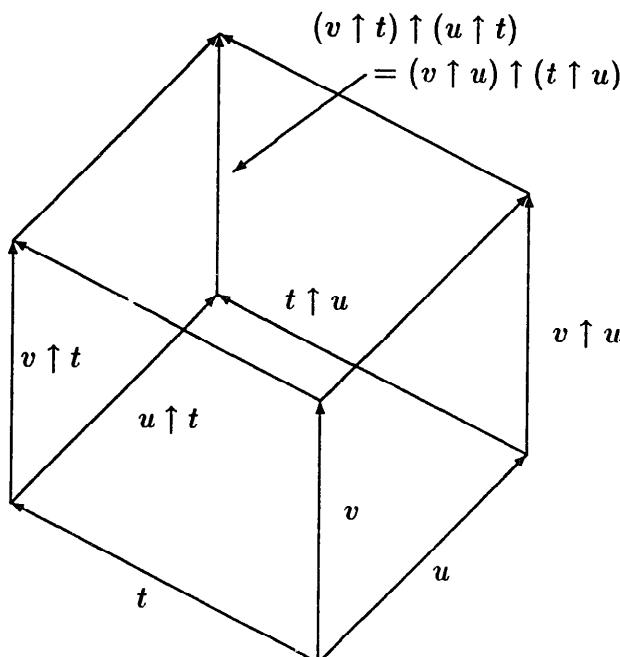


Fig. 1. CTS cube axiom.

In the sequel, we will drop the  $\#$  from  $\text{dom}^\#$  and  $\text{cod}^\#$ . We will also drop the  $q$  from “ $\text{id}_q$ ” and “ $\omega_q$ ” when it can be inferred from the context. Note that it automatically follows from the definition of a CTS that  $\omega_q \uparrow t = \omega_r$  and  $t \uparrow \omega_q = \omega_\Omega = \text{id}_\Omega$  for all transitions  $t: q \rightarrow r$ , since for each  $q$ , the transition  $\omega_q$  is the only transition with domain  $q$  and codomain  $\Omega$ . 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*. 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  $t \uparrow v = \text{id}$  and  $v \uparrow t = u$ .

**Example 1 (CTS's from graphs).** Every graph with identities  $G$  lifts to a CTS  $\text{Cts}(G)$  in a “minimally consistent” way. More precisely, suppose  $G = (O, A, \text{dom}, \text{cod}, \text{id})$ . For coinitial arrows  $t, u$  in  $G$ , let  $t \uparrow u$  be defined as follows:

$$t \uparrow u = \begin{cases} t & \text{if } u = \text{id}, \\ \text{id} & \text{if } t = u \text{ or } t = \text{id}, \\ \omega & \text{otherwise.} \end{cases}$$

It is easily verified that  $\text{Cts}(G) = (G, \uparrow)$  is a CTS.

**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  $\leqslant$  is the prefix relation induced by the monoid operation (i.e.  $t \leqslant u$  iff  $\exists v(tv = u)$ ), then  $\leqslant$  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” [31]. Formally, a *concurrent alphabet* is a pair  $(\Sigma, \parallel)$ , where  $\Sigma$  is a set and  $\parallel$  is an irreflexive, symmetric relation on  $\Sigma$ , called a *concurrency relation*. The concurrency relation induces a congruence  $\sim$  on the free monoid  $\Sigma^*$ , 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  $\varepsilon$  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  $t \uparrow tu = \varepsilon$  and  $tu \uparrow t = u$ .

**Example 3 (CTS's from Petri nets).** In [43] 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  $\text{pre}(e)$  of *preconditions* of  $e$  is the set of all  $b \in B$  such that  $(b, e) \in F$ , and the set  $\text{post}(e)$  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  $\text{pre}(e_1) \cup \text{post}(e_1)$  and  $\text{pre}(e_2) \cup \text{post}(e_2)$  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  $e \in u$ ,  $\text{pre}(e) \subseteq c$ ,
- (2) for all  $e \in u$ ,  $\text{post}(e) \cap c' = \emptyset$ ,
- (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 can 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  $\text{dom}(c, u, c') = c$  and  $\text{cod}(c, u, c') = c'$ , 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

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

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

**Example 4 (CTS's from  $\lambda$ -calculus).** Let  $\Lambda$  be the set of terms of the pure  $\lambda$ -calculus [4], and let  $\rightarrow$  denote reduction with respect to rule  $(\beta)$ . If  $C$  is a set of redexes in a term  $q$ , then a *derivation relative* to  $C$  is a derivation  $q = q_0 \rightarrow q_1 \rightarrow \dots \rightarrow q_n$ , 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  $q \rightarrow^C r$ , if  $C$  is a set of redexes in  $q$  and any complete derivation relative to  $C$  results in  $r$ . Let us call such triples  $q \rightarrow^C r$  *transitions*. Suppose  $t$  is a transition  $q \rightarrow^C r$  and  $u$  is a transition  $q \rightarrow^D s$ . It can be shown that the same set  $C \cap D$  of residuals of redexes in  $C$  is obtained for all complete derivations  $d$  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  $s \rightarrow^{C/D} p$ , where  $p$  is the unique result of a complete derivation from  $s$  relative to  $C/D$ .

We now define a pre-CTS whose state set is  $\Lambda$ , and whose transitions are all transitions  $t$  as defined above. The domain  $\text{dom}(t)$  of a transition  $t: q \rightarrow^C r$  is the term  $q$  and the codomain  $\text{cod}(t)$  of  $t$  is the term  $r$ . The identity transitions are those

of the form  $q \rightarrow^\theta q$ , all coinitial pairs of proper transitions are consistent, and we define  $t \uparrow u = t/u$ . That these definitions satisfy the axioms for a pre-CTS follows from results of Lévy [6, 27]. Although CTS axiom (4) is not satisfied, we can obtain a CTS by identifying coinitial transitions  $t$  and  $u$  whenever  $t \uparrow u = \text{id}$  and  $u \uparrow t = \text{id}$ . A similar construction can be used to obtain CTS's from left-linear term-rewriting systems without critical pairs [18].

### 2.1. Consequences of the axioms

Define a relation  $\leqslant$  on the transitions of a CTS by  $t \leqslant u$  iff  $t, u$  are coinitial and  $t \uparrow u = \text{id}$ . We call  $\leqslant$  the *prefix relation*.

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

**Proof.** Reflexivity holds because  $t \uparrow t = \text{id}$  by axiom (2c).

To show transitivity, suppose  $t \leqslant u$  and  $u \leqslant v$ . Then  $t \uparrow u = \text{id}$ , so  $(t \uparrow u) \uparrow (v \uparrow u) = \text{id}$  by axiom (2a). Since  $(t \uparrow u) \uparrow (v \uparrow u) = (t \uparrow v) \uparrow (u \uparrow v)$  by axiom (3), it follows that  $(t \uparrow v) \uparrow (u \uparrow v) = \text{id}$ . But  $u \uparrow v = \text{id}$  because  $u \leqslant v$ , hence  $t \uparrow v = \text{id}$  by axiom (2b).

Finally,  $\leqslant$  is antisymmetric because if  $t \leqslant u$  and  $u \leqslant t$ , then  $t \uparrow u$  and  $u \uparrow t$  are identities, hence  $t = u$  by axiom (4).  $\square$

**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  $t \uparrow v' = \text{id}$ , so  $v \uparrow v' = (v \uparrow v') \uparrow (t \uparrow v')$  by axiom (2)(b). By axiom (3), this is equal to  $(v \uparrow t) \uparrow (v' \uparrow t) = u \uparrow u$ , which is an identity by axiom (2)(c). A symmetric argument shows that  $v' \uparrow v = \text{id}$ , so  $v = v'$  by axiom (4).  $\square$

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 Fig. 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  $(t \uparrow v) \uparrow (tu \uparrow v) = (t \uparrow tu) \uparrow (v \uparrow tu) = \text{id}$ , and  $(tu \uparrow v) \uparrow (t \uparrow v) = (tu \uparrow t) \uparrow (v \uparrow t) = u \uparrow (v \uparrow t)$ , the result follows.  $\square$

**Lemma 2.1.4.** *Composition obeys the following laws:*

- (1) *For all  $t$ ,  $t = \text{id}$   $t = t \text{id}$ .*
- (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  $u = v$ .*

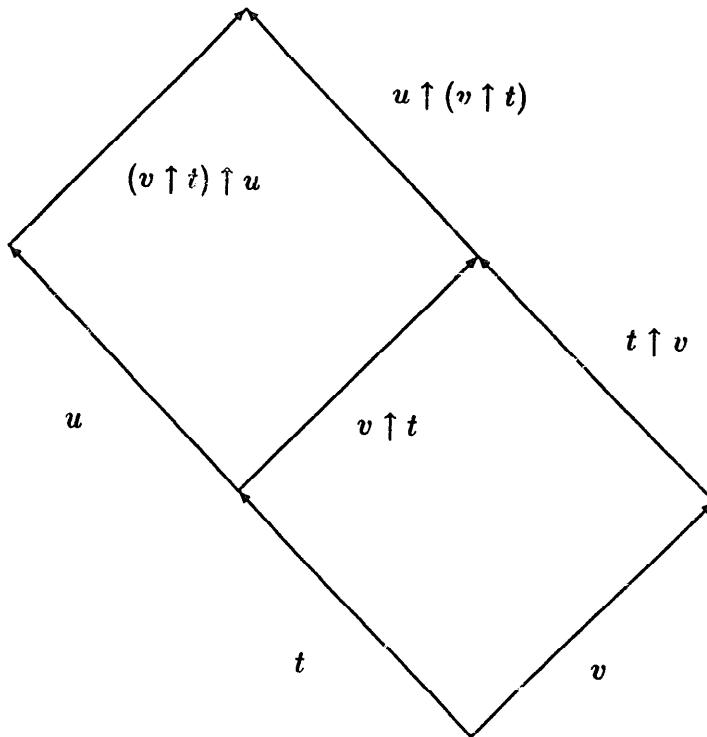


Fig. 2. Properties of the residual operation.

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

To show (2)(a), 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  $t \uparrow tu = \text{id}$ , so  $(tu)v \uparrow t = uv$ . Since by Lemma 2.1.3,

$$t \uparrow (tu)v = (t \uparrow tu) \uparrow v = \text{id},$$

it follows that  $(tu)v = t(uv)$ .

To show (2)(b), 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

$$u \uparrow v = (tu \uparrow t) \uparrow (tv \uparrow t) = (tu \uparrow tv) \uparrow (t \uparrow tv) = \text{id}.$$

Similarly,  $v \uparrow u = \text{id}$ , so  $u = v$ .  $\square$

We say that a transition  $v$  is a *join* of the coinitial transitions  $t$ ,  $u$  if  $t \leq v$ ,  $u \leq v$ ,  $v \uparrow t = u \uparrow t$ , 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  $t \uparrow v = \text{id}$  and  $v \uparrow t = u \uparrow t$ , so  $v = t(u \uparrow t)$ . Conversely, if  $v = t(u \uparrow t)$ , then  $t \leq v$ ,  $v \uparrow t = u \uparrow t$ , and

$$u \uparrow v = (u \uparrow t) \uparrow (u \uparrow t) = \text{id},$$

so  $u \leq v$ , 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  $u$ .  $\square$

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  $\leq$ .*

**Proof.** By definition,  $t \vee u$  is an upper bound of  $t$  and  $u$  under  $\leq$ . Suppose  $l$  is any upper bound of  $t$  and  $u$ . Let  $v = ! \uparrow t$  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  $u \uparrow t \leq v$ . 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  $t \vee u \leq l$ .  $\square$

## 2.2. The category CTS

If  $G = (O, A, \text{dom}, \text{cod}, \text{id})$  and  $G' = (O', A', \text{dom}', \text{cod}', \text{id}')$  are graphs with identities, then a *graph morphism* from  $G$  to  $G'$  is a pair of maps  $\rho = (\rho_o, \rho_a)$ , where  $\rho_o: O \rightarrow O'$  and  $\rho_a: A \rightarrow A'$ , such that  $\text{dom}' \circ \rho_a = \rho_o \circ \text{dom}$ ,  $\text{cod}' \circ \rho_a = \rho_o \circ \text{cod}$ , and  $\rho_a \circ \text{id} = \text{id}' \circ \rho_o$ . In the sequel, we will drop the notational distinction between  $\rho_o$  and  $\rho_a$ , writing simply  $\rho$  in both cases. Let **Graph** denote the category of graphs and their morphisms.

### 2.2.1. CTS-morphisms

A *CTS-morphism* from a CTS  $C = (G, \uparrow)$  to a CTS  $C' = (G', \uparrow')$  is a graph morphism  $\rho: G \rightarrow G'$ , with the following property:

- 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  $\rho: C \rightarrow C'$  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 class 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  $\rho: C \rightarrow C'$  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  $\rho(t \vee u) = \rho(t) \vee \rho(u)$ .  $\square$

Some CTS's to which we shall frequently refer are **0** (the CTS with no proper states) and **1** (the CTS with one proper state and one proper transition). It is not difficult to see that **0** is an initial object in CTS and that **1** is a terminal object.

**Lemma 2.2.2.** (1) *A morphism  $\mu : C \rightarrow D$  is an isomorphism iff it is bijective on transitions, and reflects consistent pairs of transitions.*

(2) *A morphism  $\mu : C \rightarrow D$  is a monomorphism iff it is injective on transitions.*

**Proof.** Straightforward.  $\square$

We have not been able to obtain a similar characterization of the epimorphisms in CTS.

### 2.2.2. Limits and coproducts

**Lemma 2.2.3.** *The forgetful functor  $\text{Graph} : \text{CTS} \rightarrow \text{Graph}$  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  $\text{Cts}(G)$ .*

**Proof.** Straightforward.  $\square$

Since it is easily seen that the functor  $\text{Cts}$  is full, faithful, and injective on objects, it follows by the preceding lemma that it is an isomorphism from  $\text{Graph}$  to a full and coreflective subcategory of CTS. Thus, any limits existing in CTS, since they are preserved by the right adjoint  $\text{Graph}$ , must be lifted from corresponding limits in  $\text{Graph}$ . This is important for our computational intuition, since it means that the computational behavior of CTS's (less familiar objects) obtained by limit constructions can be understood in terms of the same constructions performed on their underlying graphs (more familiar objects).

**Theorem 2.1.** CTS is small complete.

**Proof.** It suffices to show that small products and equalizers of pairs of morphisms lift from  $\text{Graph}$  to CTS, since a standard category-theoretic result (see, e.g. [28, p.

109]) states that the existence of small products and equalizers of pairs of morphisms implies the existence of all small limits. Lifting products from **Graph** to **CTS** is accomplished by the obvious “componentwise” construction. A straightforward “restriction of structure” construction suffices in the case of equalizers.  $\square$

**Theorem 2.2.** ***CTS** has all small coproducts, which are preserved by the functor **Graph**.*

**Proof.** Coproducts in **CTS** are obtained by taking the coproduct (disjoint union) of the underlying graphs, and defining  $t, u$  to be consistent only when both transitions are in the same summand, and are consistent in that summand.  $\square$

We know very little about the existence or nonexistence of more general colimits in **CTS**. Because of the “cube axiom,” it seems difficult to find a useful general theorem about quotient constructions on **CTS**’s.

### 2.2.3. Cartesian closure

Surprisingly, the category **CTS** is cartesian closed. The construction involves an interesting notion of “natural translation” between morphisms, which is similar to the notion of “natural transformation” between functors.

Formally, suppose  $\sigma, \rho: C \rightarrow D$  are morphisms. A *natural translation* from  $\sigma$  to  $\rho$  consists of a collection of transitions  $\mathcal{U} = \{\mathcal{U}_q : q \in C\}$  of  $D$ , such that for each transition  $t: q \rightarrow r$  in  $C$ , the following “naturality conditions” hold:

- (1)  $\mathcal{U}_q \uparrow \sigma(t) = \mathcal{U}_r$ .
- (2)  $\sigma(t) \uparrow \mathcal{U}_q = \rho(t)$ .
- (3) The join  $\sigma(t) \vee \mathcal{U}_q$  exists.

The transitions  $\mathcal{U}_q$  are called the *components* of  $\mathcal{U}$ . If  $\mathcal{U}$  is a natural translation from  $\sigma$  to  $\rho$ , then we write  $\mathcal{U}: \sigma \rightarrow \rho$ .

Suppose  $\mathcal{U}: \sigma \rightarrow \rho$  and  $\mathcal{V}: \sigma \rightarrow \tau$  are natural translations, where  $\sigma, \rho, \tau: C \rightarrow D$ . Then  $\mathcal{U}$  and  $\mathcal{V}$  are *consistent* if for each state  $q$  of  $C$ , the components  $\mathcal{U}_q$  and  $\mathcal{V}_q$  are consistent. If  $\mathcal{U}$  and  $\mathcal{V}$  are consistent, then let  $\delta: C \rightarrow D$  be defined by  $\delta(t) = \rho(t) \uparrow (\mathcal{V}_q \uparrow \mathcal{U}_q)$  (equivalently,  $\delta(t) = \tau(t) \uparrow (\mathcal{U}_q \uparrow \mathcal{V}_q)$ ). Define the *residual* of  $\mathcal{U}$  after  $\mathcal{V}$  by  $(\mathcal{U} \uparrow \mathcal{V})_q = \mathcal{U}_q \uparrow \mathcal{V}_q$ , which is easily seen to be a natural translation from  $\tau$  to  $\delta$ .

Given **CTS**’s  $C$  and  $D$ , construct the “exponential” **CTS**  $D^C$  as follows: Let the states of  $D^C$  be the morphisms from  $C$  to  $D$ , and, for each pair of morphisms  $\sigma, \tau: C \rightarrow D$ , let the transitions from  $\sigma$  to  $\tau$  in  $D^C$  be all natural translations from  $\sigma$  to  $\tau$ . Let the identities of  $D^C$  be the natural translations with each component an identity transition, and let consistency and residual be defined as in the previous paragraph. It is straightforward to check that  $D^C$  satisfies the **CTS** axioms.

**Theorem 2.3.** *For each **CTS**  $C$ , the map taking  $D$  to  $D^C$  is the object map of a functor  $(-)^C: \mathbf{CTS} \rightarrow \mathbf{CTS}$ , which is right adjoint to the functor  $(-\times C)$  taking  $D$  to  $D \times C$ .*

**Proof.** It suffices to show, for each  $D$ , the existence of an “evaluation map”  $\eta_D: D^C \times C \rightarrow D$ , universal from  $(-\times C)$  to  $D$ . The definition of  $\eta_D$  is as follows:

If  $\mathcal{U}$  is a transition of  $D^C$ , and  $t$  is a transition of  $C$ , then define  $\eta_D(\mathcal{U}, t) = \mathcal{U}_{\text{dom}(t)} \vee \text{dom}(\mathcal{U})(t)$ .

We first verify that  $\eta_D$  is a morphism. Suppose  $(\mathcal{U}, t)$  and  $(\mathcal{V}, u)$  are consistent transitions of  $D^C \times C$ , where  $\mathcal{U}: \sigma \hookrightarrow \rho$  and  $\mathcal{V}: \sigma \hookrightarrow \tau$ . Then

$$\begin{aligned}\eta_D((\mathcal{U}, t) \uparrow (\mathcal{V}, u)) &= \eta_D(\mathcal{U} \uparrow \mathcal{V}, t \uparrow u) \\ &= (\mathcal{U}_{\text{cod}(u)} \uparrow \mathcal{V}_{\text{cod}(u)}) \vee (\tau(t) \uparrow \tau(u)) \\ &= (\mathcal{U}_{\text{cod}(u)} \vee (\sigma(t) \uparrow \sigma(u))) \uparrow \mathcal{V}_{\text{cod}(u)} \\ &= ((\mathcal{U}_{\text{dom}(u)} \vee \sigma(t)) \uparrow \sigma(u)) \uparrow \mathcal{V}_{\text{cod}(u)} \\ &= (\mathcal{U}_{\text{dom}(t)} \vee \sigma(t)) \uparrow (\mathcal{V}_{\text{dom}(u)} \vee \sigma(u)) \\ &= \eta_D(\mathcal{U}, t) \uparrow \eta_D(\mathcal{V}, u),\end{aligned}$$

where we have made extensive use of the defining “naturality” properties of natural translations.

Finally, we show that  $\eta_D$  is universal from  $(-\times C)$  to  $D$ . Suppose  $\tau: B \times C \rightarrow D$  is a morphism. Let  $\mu: B \rightarrow D^C$  take each state  $q$  of  $B$  to the morphism  $\tau(q, -): C \rightarrow D$ , and each transition  $u: q \rightarrow r$  of  $B$  to the natural translation  $\mathcal{U}: \tau(q, -) \hookrightarrow \tau(r, -)$  with components  $\mathcal{U}_p = \tau(u, p)$ . Note that  $\tau = \eta_D \circ (\mu \times 1_C)$ , since if  $t: p \rightarrow s$  is a transition of  $C$  and  $u: q \rightarrow r$  is a transition of  $B$ , then

$$\eta_D(\mu(u), t) = (\mu(u))_p \vee \text{dom}(\mu(u))(t) = \tau(u, p) \vee \tau(q, t) = \tau(u, t).$$

Moreover, the condition  $\tau = \eta_D \circ (\mu \times 1_C)$  uniquely determines  $\mu$ , since for each transition  $u: q \rightarrow r$  of  $B$  and state  $p$  of  $C$  we must have

$$\tau(u, p) = \eta_D(\mu(u), p) = (\mu(u))_p \vee \text{dom}(\mu(u))(p) = (\mu(u))_p. \quad \square$$

### 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.

Note that properties (1) and (3) imply that  $C$  has a unique terminal object  $\Omega$ . Moreover,  $C$  contains no arrow  $t: \Omega \rightarrow q$  for  $q \neq \Omega$ . To see this, suppose there were such a  $t$ . Then  $t\omega_q = \text{id}_\Omega$ , because  $\Omega$  is a terminal object, hence  $t = (t\omega_q)t$ . But  $(t\omega_q)t = t(\omega_q t)$ , hence  $\omega_q t = \text{id}_q$  by the fact that  $t$  is an epimorphism. Thus,  $t$  is an isomorphism from  $\Omega$  to  $q$ , contradicting the uniqueness of  $\Omega$ . Because of these facts, if  $C = (G, \cdot)$  is a computation category, then we may regard  $G$  as an augmented graph  $G^*$ .

**Theorem 2.4.** Suppose  $C = (G, \uparrow)$  is a complete CTS, and let  $\cdot$  denote the composition operation of  $C$ . Then  $C' = (G^*, \cdot)$  is a computation category. Conversely, suppose  $C' = (G', \cdot)$  is a computation category. Regard  $G'$  as an augmented graph  $G^*$ . For coinitial arrows  $t, u$ , let  $t \uparrow u$  denote the arrow opposite  $t$  in the pushout square determined by  $t$  and  $u$ . Then  $C = (G, \uparrow)$  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  $(G, \uparrow)$  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  $\Omega$  is clearly a terminal object of  $C'$ . If  $v$  is an isomorphism of  $C'$ , with inverse  $v'$ , then  $v \leq v v' = \text{id}$  and  $\text{id} \leq v$ , so  $v = \text{id} = v'$  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  $C'$ .  $\square$

Let **CCTS** denote the full subcategory of **CTS** whose objects are the complete CTS's, and let **CCts**: **CCTS**  $\rightarrow$  **CTS** denote the inclusion functor. We now show that **CCts** has a left adjoint; that is, every CTS  $C$  freely generates a complete CTS  $C^*$ , having 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 [27] in the setting of the  $\lambda$ -calculus, and adapted in [6, 18] to the cases of recursive programs and left-linear term-rewriting systems without critical pairs. Consideration of problems in the theory of concurrency led to its independent rediscovery by the author in the present axiomatic setting.

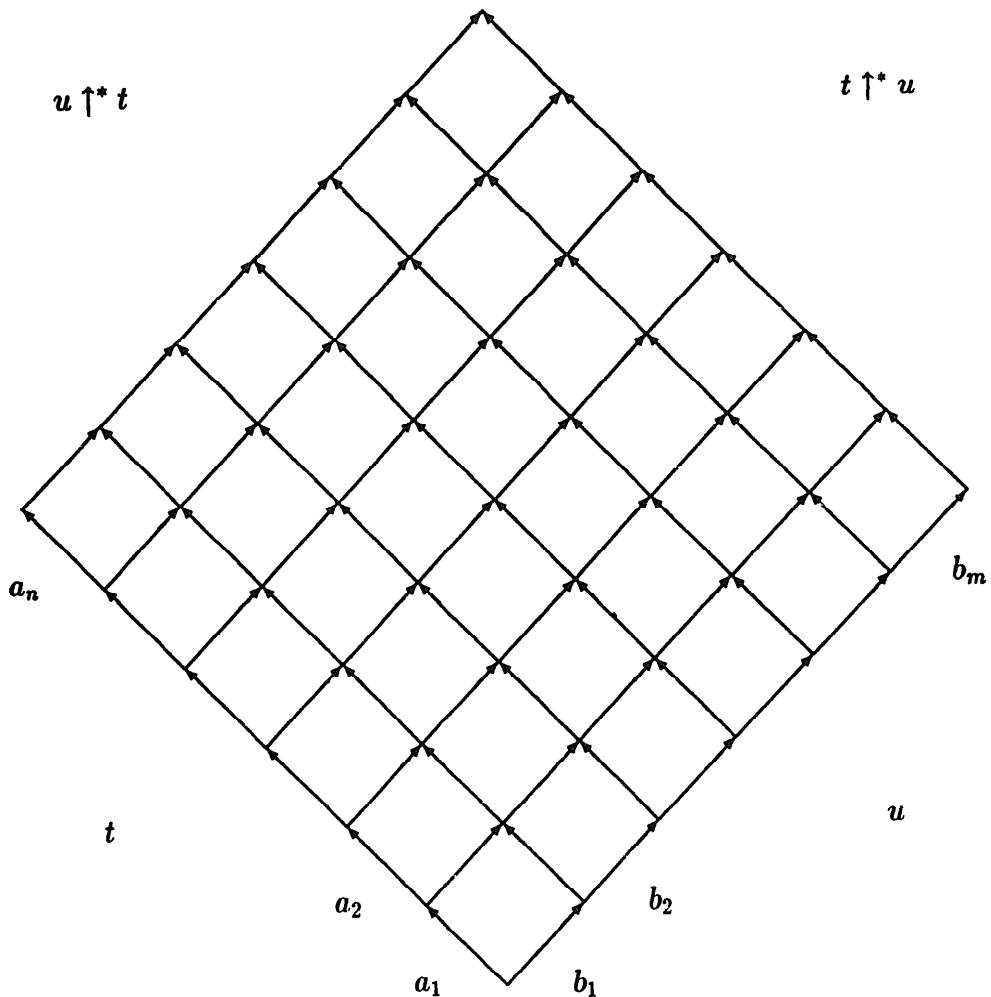
Suppose  $C = (G, \uparrow)$  is a CTS. Let  $G^*$  denote the free category (objects = objects of  $G$ , identities = identities of  $G$ , nonidentity arrows = finite composable sequences of nonidentity arrows of  $G$ ) generated by  $G$ . Let  $\uparrow^*$  be the extension to  $(G^*)^*$  of the operation  $\uparrow$  on  $G^*$ , defined recursively by the following properties (see Fig. 3):

- (1) For all  $t: q \rightarrow r$  in  $G^*$ , define  $\text{id}_q \uparrow^* t = \text{id}_r$ , and  $t \uparrow^* \text{id}_q = t$ .
- (2) For all  $t: q \rightarrow r$  in  $(G^*)^*$ , define  $\omega_q \uparrow^* t = \omega_r$ , and  $t \uparrow^* \omega_q = \omega_\Omega$ .
- (3) For all  $t$  in  $G^*$  and all  $a, b \in G$ , with  $a, b$  coinitial and  $a, t$  composable,

$$at \uparrow^* b = \begin{cases} (a \uparrow b)(t \uparrow^*(b \uparrow a)), & \text{if } a \uparrow b \neq \omega \text{ and } t \uparrow^*(b \uparrow a) \neq \omega, \\ \omega, & \text{otherwise.} \end{cases}$$

- (4) For all  $t, u$  in  $G^*$  with  $u$  not in  $G$ , and all  $b \in G$ , with  $b, t$  coinitial and  $b, u$  composable,

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

Fig. 3. Extension of  $\uparrow$  to  $\uparrow^*$ .

**Lemma 2.3.1.** *The structure  $(G^*, \uparrow^*)$  is a pre-CTS that satisfies the following additional property: For all transitions  $t, u, v$ , with  $t$  and  $u$  coinitial, and  $u$  and  $v$  composable,*

- (1)  $t\uparrow^*uv = (t\uparrow^*u)\uparrow^*v$ ,
- (2)  $uv\uparrow^*t = (u\uparrow^*t)(v\uparrow^*(t\uparrow^*u))$ .

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

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

**Lemma 2.3.2.** *The relation  $\sim$  has the following properties:*

- (1)  $\sim$  is an equivalence relation.
- (2) For all arrows  $t, u$  of  $G^*$ , if  $t \sim u$ , then  $\text{dom}(t) = \text{dom}(u)$  and  $\text{cod}(t) = \text{cod}(u)$ .
- (3) For all arrows  $t, t', u, u'$  of  $G^*$ , if  $t \sim t'$ ,  $u \sim u'$ , and  $t, u$  are consistent, then  $t', u'$  are consistent, and  $t\uparrow^*u \sim t'\uparrow^*u'$ .
- (4) For all coinitial arrows  $t, u$  of  $G^*$ , if  $t\uparrow^*u \sim \text{id}$  and  $u\uparrow^*t \sim \text{id}$ , then  $t \sim u$ .

**Proof.** (1): It is clear that  $\sim$  is symmetric. It is reflexive because  $t \uparrow^* t = \text{id}$  by Lemma 2.3.1. Transitivity follows from Lemma 2.3.1, as in the proof of Lemma 2.1.1.

(2) is obvious from Lemma 2.3.1 and the definition of  $\sim$ .

(3): Suppose transitions  $t, t', u, u'$  of  $\text{Cts}(G^*)$  are such that  $t \sim t'$ ,  $u \sim u'$ , and  $t, u$  are consistent. By Lemma 2.3.1 and the assumption that  $t \sim t'$ , we have that  $(t' \uparrow^* t) \uparrow^* (u \uparrow^* t) = \text{id}$ . But

$$(t' \uparrow^* t) \uparrow^* (u \uparrow^* t) = (t' \uparrow^* u) \uparrow^* (t \uparrow^* u)$$

by Lemma 2.3.1, thus  $(t' \uparrow^* u) \uparrow^* (t \uparrow^* u) = \text{id}$ . Similarly,

$$(t \uparrow^* u) \uparrow^* (t' \uparrow^* u) = (t \uparrow^* t') \uparrow^* (u \uparrow^* t') = \text{id},$$

hence  $t \uparrow^* u \sim t' \uparrow^* u$ . Since  $t$  and  $u$  are consistent by assumption, it follows immediately that  $t'$  and  $u$  are consistent. Now,  $u \uparrow^* u = \text{id}$ , so  $t' \uparrow^* u = (t' \uparrow^* u) \uparrow^* (u \uparrow^* u)$  by Lemma 2.3.1. Since

$$(t' \uparrow^* u) \uparrow^* (u' \uparrow^* u) = (t' \uparrow^* u') \uparrow^* (u \uparrow^* u')$$

by Lemma 2.3.1, and since  $u \uparrow^* u = \text{id}$ , it follows that  $t'$  and  $u'$  are consistent, and  $t' \uparrow^* u = t' \uparrow^* u'$  by Lemma 2.3.1. Thus,  $t \uparrow^* u \sim t' \uparrow^* u = t' \uparrow^* u'$ .

(4): First note that it is immediate from Lemma 2.3.1 that  $t \sim \text{id}_q$  holds for a transition of  $\text{Cts}(G^*)$  iff  $t = \text{id}_q$ . Thus, if  $t \uparrow^* u \sim \text{id}$  and  $u \uparrow^* t \sim \text{id}$ , then  $t \uparrow^* u = \text{id}$  and  $u \uparrow^* t = \text{id}$ . But this states exactly that  $t \sim u$ .  $\square$

We now form the CTS  $C^*$  as follows:

- Take the proper states of  $C$  as the proper states of  $C^*$ .
- Define the proper transitions of  $C^*$  to be the  $\sim$ -equivalence classes. We write  $[t]$  for the equivalence class of  $t$ , and we define the identities of  $C^*$  to be the equivalence classes of identities of  $C$ . Define  $\text{dom}([t]) = \text{dom}(t)$  and  $\text{cod}([t]) = \text{cod}(t)$ .
- 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 Lemma 2.3.2, that these definitions are independent of the choice of  $t$  and  $u$ . It is also straightforward to check that  $C/\sim$  satisfies the CTS axioms. Moreover,  $C^*$  is complete, since if  $[t]$  and  $[u]$  are composable transitions of  $C^*$ , then  $[t] \uparrow [tu] = [t \uparrow^* tu] = [\text{id}]$  and  $[t][u] \uparrow [t] = [tu \uparrow^* t] = [u]$  by Lemma 2.3.1. Hence  $[tu] = [t][u]$ .

**Theorem 2.5.** *The map, taking each CTS  $C$  to its completion  $C^*$ , is the object map of a functor  $\text{Cts} : \text{CTS} \rightarrow \text{CCTS}$ , which is left-adjoint to the inclusion of  $\text{CCTS}$  in  $\text{CTS}$ .*

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

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

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

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 of the form  $[a_1 a_2 \dots a_n]$ , where  $n > 0$  and each  $a_i$  is a proper transition of  $C$ . By Lemma 2.2.1, any morphism  $\rho^*: C^* \rightarrow D$  must satisfy

$$\rho^*([a_1 a_2 \dots a_n]) = \rho^*([a_1])\rho^*([a_2]) \dots \rho^*([a_n]).$$

The condition  $\rho^* \circ \mu_C = \rho$  implies in addition that  $\rho^*([a_i]) = \rho(a_i)$ , hence

$$\rho^*([a_1 a_2 \dots a_n]) = \rho(a_1)\rho(a_2) \dots \rho(a_n).$$

Thus, there can be at most one morphism  $\rho^*: C^* \rightarrow D$  satisfying  $\rho^* \circ \mu_C = \rho$ .

To show that  $\rho^*$  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  $[a_1 a_2 \dots a_n] = [b_1 b_2 \dots b_m]$ , then

$$\rho(a_1)\rho(a_2) \dots \rho(a_n) = \rho(b_1)\rho(b_2) \dots \rho(b_m).$$

But if  $[a_1 a_2 \dots a_n] = [b_1 b_2 \dots b_m]$ , then

$$a_1 a_2 \dots a_n \uparrow^* b_1 b_2 \dots b_m = \text{id}^n = \text{id}$$

and

$$b_1 b_2 \dots b_m \uparrow^* a_1 a_2 \dots a_n = \text{id}^m = \text{id}.$$

Applying the fact noted above, we see that

$$\rho(a_1)\rho(a_2) \dots \rho(a_n) \uparrow \rho(b_1)\rho(b_2) \dots \rho(b_m)$$

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  $\rho^*$  is a morphism, we must show that it preserves identities and residuals of consistent pairs of transitions. Each identity of  $C^*$  is of the form  $[\text{id}_q]$ , where  $\text{id}_q$  is an identity of  $C$ , and we must have  $\rho^*([\text{id}_q]) = \rho(\text{id}_q) = \text{id}_q$ . Moreover, if  $[a_1 a_2 \dots a_n]$  and  $[b_1 b_2 \dots b_m]$  are consistent, then using the fact noted above shows that

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

as required.  $\square$

We often need to know which transitions of  $C^*$  are *atomic*, in the sense that they correspond to single transitions of  $C$ . In contrast to the case of ordinary NDTS's, where the structure of a free category is sufficient to recover the generating graph, we cannot uniquely recover a CTS  $C$  from its completion  $C^*$ . Thus, we define the *subobject of atoms* of  $C^*$  to be the “inclusion of generators”  $\mu_C : C \rightarrow C^*$ . A transition of  $C^*$  is called an *atom* if it is  $\mu_C(t)$  for some transition  $t$  of  $C$ .

## 2.4. Computation diagrams

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

A *pointed CTS* is an object of the comma category  $(1 \downarrow \text{CTS})$ , where  $1$  is a terminal object in CTS. That is, a pointed CTS is a pair  $(C, q_*)$ , where  $C$  is a CTS, and  $q_*$  is a state of  $C$ , called the *start state*.

A *computation diagram* is a pointed CTS  $(D, \perp)$ , such that  $\perp$  is an initial object of the completion  $D^*$  of  $D$ . It is easy to see then that the prefix relation on the initial arrows of  $D^*$  determines a corresponding ordering on the states of  $D^*$ , showing that  $D^*$  is a join-semilattice with  $\perp$  as least element and  $\Omega$  as greatest element. A *complete computation diagram* is a computation diagram  $(D, \perp)$  such that  $D$  is complete. Let  $\text{CDiag}$  and  $\text{CCDiag}$  denote the full subcategories of  $(1 \downarrow \text{CTS})$  whose objects are the computation diagrams, and the complete computation diagrams, respectively.

Each pointed CTS  $(C, q_*)$  determines a corresponding complete computation diagram  $(q_* \downarrow C^*)$ , which we call the *complete computation diagram* of  $(C, q_*)$ . The objects of  $(q_* \downarrow C^*)$  are called the *events* of  $(C, q_*)$ . The transitions of  $(q_* \downarrow C^*)$  are pairs  $(t, u)$  of events of  $C^*$  such that  $t \leq u$ . A set  $J$  of events of  $(C, q_*)$  is called *consistent* if the join of each finite subset of  $J$  is a proper event.

**Theorem 2.6.** *The map, taking each pointed, complete CTS  $(C, q_*)$  to its complete computation diagram  $(q_* \downarrow C^*)$ , is the object map of a functor, which is right-adjoint to the inclusion of  $\text{CCDiag}$  in  $(1 \downarrow \text{CTS})$ .*

**Proof.** Let the “natural map”  $\eta_C : (q_* \downarrow C^*) \rightarrow (C, q_*)$  take each transition  $(t, u)$  of  $(q_* \downarrow C^*)$  to the corresponding transition  $u \uparrow t$  of  $(C^*, q_*)$ , which we may regard as a transition of  $(C, q_*)$  because  $C$  is complete, therefore isomorphic to  $C^*$ . It is straightforward to check that  $\eta_C$  is a pointed-CTS morphism, and that every pointed-CTS morphism  $\rho$  from a complete computation diagram to  $(C, q_*)$  factors uniquely through  $\eta_C$ .  $\square$

The *computation diagram of a pointed CTS*  $(C, q_i)$  is the subobject  $\sigma : (D, \perp) \rightarrow (q_i \downarrow C^*)$  making the following diagram a pullback in  $(1 \downarrow \text{CTS})$ :

$$\begin{array}{ccc} (C, q_i) & \xrightarrow{\mu_C} & (C^*, q_i) \\ \uparrow & & \uparrow \eta_{C^*} \\ (D, \perp) & \xrightarrow{\sigma} & (q_i \downarrow C^*) \end{array}$$

Here  $\mu_C : (C, q_i) \rightarrow (C^*, q_i)$  is the “inclusion of generators,” and  $\eta_{C^*} : (q_i \downarrow C^*) \rightarrow (C^*, q_i)$  is the “natural map” of the previous theorem. It is not difficult to see that  $(D^*, \perp) \simeq (q_i \downarrow C^*)$ , and thus  $(D, \perp)$  may be regarded as the “subdiagram of atoms” or “Hasse diagram” of  $(q_i \downarrow C^*)$ .

**Theorem 2.7.** *The map, taking each pointed CTS to its computation diagram, is right-adjoint to the inclusion of CDiag in  $(1 \downarrow \text{CTS})$ .*

**Proof.** Straightforward.  $\square$

In view of the preceding result, we shall use the “comma category” notation  $(q_i \downarrow C)$  to denote the computation diagram  $(D, \perp)$  of  $(C, q_i)$ , even though  $C$  is not necessarily a category.

We are now ready to define the “computations” of a pointed CTS. Recall that an *ideal* of a partially ordered set  $(K, \leq)$  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  $u \leq t$ , then  $u \in J$ ;
- (3) (directed): If  $t \in J$  and  $u \in J$ , then there exists  $v \in J$  such that  $t \leq v$  and  $u \leq v$ .

An ideal  $J$  of  $K$  is *principal* if it is the set of all elements below some element  $t$  of  $K$ . It is *proper* if  $J \neq K$ .

- A *computation* of a pointed CTS  $(C, q_i)$  is a proper ideal of its complete computation diagram  $(q_i \downarrow C^*)$ . A computation is *finite* if it is a principal ideal, otherwise it is *infinite*.

The preceding definition of computation has some convenient properties, which we summarize below. A set  $\mathcal{J}$  of computations of  $(C, q_i)$  is called *chain-complete* if whenever  $\mathcal{J}'$  is a subset of  $\mathcal{J}$  that is linearly ordered with respect to inclusion, then  $\bigcup \mathcal{J}'$  is also an element of  $\mathcal{J}$ .

### Lemma 2.4.1

- (1) *The set of computations of a pointed CTS  $(C, q_i)$  is an algebraic directed-complete poset under inclusion order, whose finite elements are exactly the principal ideals.*
- (2) *Every consistent set of events of  $(C, q_i)$  is included in a unique least computation.*
- (3) *If  $\mathcal{J}$  is a nonempty, chain-complete set of computations of  $(C, q_i)$ , then every element of  $\mathcal{J}$  is included in a maximal element of  $\mathcal{J}$ .*

**Proof.** (1): That the set of computations of  $(C, q_i)$  is an algebraic directed-complete poset, with the principal ideals as finite elements, is a standard property of the “completion by ideals” of a partially ordered set (see e.g. [14]).

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

(3) is just Zorn’s Lemma applied to sets of computations.  $\square$

In view of this result, we henceforth identify the events of  $(C, q_i)$  with the corresponding finite computations. We say that a collection  $\mathcal{J}$  of computations is *consistent* if  $\bigcup \mathcal{J}$  is consistent, in which case we define  $\bigvee \mathcal{J}$  to be the least computation containing  $\bigcup \mathcal{J}$ .

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

A *computation sequence* for a pointed CTS  $(C, q_i)$  is a finite or infinite sequence

$$t_0 \leq t_1 \leq t_2 \leq \dots$$

of events of  $(C, q_i)$ , such that  $t_0 = \text{id}_{q_i}$ , and for each  $k \geq 0$ , the transition  $t_{k+1} \uparrow t_k$  is an atom (possibly an identity). The ideal  $\bigvee_k t_k$  is called the *computation generated* by the sequence  $t_0 \leq t_1 \leq t_2 \leq \dots$ . For a finite computation sequence

$$t_0 \leq t_1 \leq \dots \leq t_n,$$

the number  $n$  is called the *length* of the sequence, and the event  $t_n$  is called its *result*.

Define the *rank* of an event  $t$  of  $(C, q_i)$  to be the least  $n$  such that  $t$  is the result of a computation sequence of length  $n$  for  $(C, q_i)$ . Since every transition  $t$  of  $C^*$  can be factored into a sequence of atoms, every event of  $(C, q_i)$  has finite rank.

**Lemma 2.4.2.** *If  $t$  is an event of  $(C, q_i)$ , then*

- (1)  *$t$  has rank 0 iff  $t = \text{id}_{q_i}$ ;*
- (2) *if  $t$  has rank  $k > 0$ , then  $t = ua$ , where  $u$  has rank  $< k$  and  $a$  is an atom.*

**Proof.** (1): If  $t = \text{id}_{q_i}$  is an identity, then  $t_0 = t$  is the required computation sequence of length 0. Conversely, if  $t$  is the result  $t_0$  of a computation sequence of length 0, then  $t = t_0$ , which equals  $\text{id}_{q_i}$  by definition of a computation sequence.

(2): Suppose  $t$  has rank  $k > 0$ . Then there exists a computation sequence

$$t_0 \leq t_1 \leq \dots \leq t_k$$

of length  $k$ , with result  $t$ . Let  $u = t_{k-1}$  and  $a = t_k \uparrow t_{k-1}$ .  $\square$

A consistent set  $T$  of transitions of a CTS  $C$  is called *dependent* if there exists  $t \in T$  such that  $t \leq \bigvee T'$  for some finite subset of  $T'$  not containing  $t$ . If  $T$  is not dependent, then it is called *independent*. A CTS  $C$  is said to have *finite concurrency* if it has no infinite independent sets of transitions. Note in particular that if  $G$  is a graph, then  $\text{Cts}(G)$  has finite concurrency.

**Theorem 2.8.** Suppose  $(C, q_i)$  is a pointed CTS, where  $C$  has finite concurrency. Then every computation of  $(C, q_i)$  is the computation generated by some computation sequence.

**Proof.** Suppose  $J$  is a computation of  $C_{q_i}$ . We construct a computation sequence

$$t_0 \leq t_1 \leq t_2 \leq \dots,$$

with each  $t_k \in J$ , inductively as follows: Define  $n_0 = 0$  and let  $t_{n_0} = \text{id}_{q_i}$ . Clearly,  $t_{n_0} \in J$ . Suppose  $t_{n_k} \in J$  has been defined, and let  $J_k = \{t \uparrow t_{n_k} : t \in J\}$ . It is easy to see that  $J_k$  is a computation of  $(C, \text{cod}(t_{n_k}))$ . Let  $U_k$  be a maximal independent set of atomic events in  $J_k$ . Since  $C$  has finite concurrency,  $U_k$  is finite, thus  $\bigvee U_k$  exists and is in  $J_k$ . Moreover,  $\bigvee U_k$  is a transition of  $C^*$ , hence can be expressed as the composition of atoms:  $a_1 a_2 \dots a_m$ . Define  $t_{k+i} = t_k a_1 \dots a_i$  for  $1 \leq i \leq m$ , and define  $n_{k+1} = n_k + m$ .

Now, we show by induction on  $k \geq 0$  that for all  $k \geq 0$ , if  $t \in J$  has rank  $k$ , then  $t \leq t_{n_k}$ . From this it follows, because every event  $t \in J$  has finite rank, that the computation sequence

$$t_0 \leq t_1 \leq t_2 \leq \dots$$

generates  $J$ . For  $k = 0$ , the event  $\text{id}_{q_i}$  is the only one of rank 0, and  $\text{id}_{q_i} = t_{n_0}$  by definition. Suppose, for some  $k \geq 0$ , that if  $u \in J$  has rank  $k$ , then  $u \leq t_{n_k}$ . Let  $t$  be of rank  $k+1$ . Then  $t = ua$ , where  $u$  is of rank  $k$  and  $a$  is an atom. By induction hypothesis,  $u \leq t_{n_k}$ . Since  $t \uparrow t_{n_k} \in J_k$ , and

$$t \uparrow t_{n_k} = (u \uparrow t_{n_k})(a \uparrow (t_{n_k} \uparrow u)) = a \uparrow (t_{n_k} \uparrow u)$$

is an atom, it must be that  $t \uparrow t_{n_k} \leq \bigvee U_k$ , hence  $t \leq t_{n_{k+1}}$ .  $\square$

### 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 [7, 13, 19, 20, 29, 40], 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. Usually, 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 receiver for 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.

For this section, some additional notation will be convenient. If  $t$  and  $u$  are proper transitions of CTS’s  $C$  and  $D$ , respectively, then we write  $t : u$  to denote the transition of  $C \times D$  whose first component is  $t$  and whose second component is  $u$ . Similarly, if  $\lambda : C \rightarrow D$  and  $\lambda' : C \rightarrow D'$  are morphisms, then  $(\lambda : \lambda') : C \rightarrow D \times D'$  maps a proper transition  $t$  of  $C$  to  $\lambda(t) : \lambda'(t)$  in  $D \times D'$ . We use  $\pi_C$  and  $\pi_D$  to denote the projections from the product  $C \times D$  to  $C$  and  $D$ , respectively. If  $\lambda : B \rightarrow C \times D$  is a morphism, then we often write  $\lambda_C$  and  $\lambda_D$  as abbreviations for  $\pi_C \circ \lambda$  and  $\pi_D \circ \lambda$ , respectively.

We will be making frequent use of trace algebras (see Section 2, Example 2). If  $X$  is a trace algebra, then we denote the identity of  $X$  by  $\varepsilon_X$ , or just  $\varepsilon$ , when  $X$  is clear from the context. We denote the prefix ordering by  $\leqslant_X$ , or just  $\leqslant$ , 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. The *ideal space* of a trace algebra  $X$  is the set  $\bar{X}$  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  $\bar{X}$ , by identifying each  $x \in X$  with the corresponding principal ideal. We can then view the inclusion order on  $\bar{X}$  as an extension of the prefix ordering  $\leqslant$  on  $X$ , and we use the same symbol  $\leqslant$  in both cases. The elements of  $X \subseteq \bar{X}$  are called the *finite* elements of  $\bar{X}$ , and all other elements of  $\bar{X}$  are called *infinite*. A morphism  $\rho : X \rightarrow Y$  extends uniquely to a continuous function  $\bar{\rho} : \bar{X} \rightarrow \bar{Y}$ . We will generally identify  $\overline{X \times Y}$  and  $\bar{X} \times \bar{Y}$ , exploiting the obvious natural isomorphism.

### 3.1. Machines

We use “machines” to model processes. The kind of machine we consider consists of two parts: a “free-running” part, which describes the behavior of the process in the absence of input, and an “input-driven” part, which describes the effect of arriving input. Our machines are obtained by using CTS’s to generalize the “sequential machines” of classical automata theory [12, 15]. In the classical case, a machine comprises a set of states  $Q$ , and an “action”  $\delta : X \rightarrow \text{Set}(Q, Q)$ , which is a monoid homomorphism from a free monoid  $X$  of input strings to the monoid  $\text{Set}(Q, Q)$  of functions from  $Q$  to  $Q$ . For each input string  $x$ , the function  $\delta(x)$  describes the change of state caused by that input. Such a machine simply responds to input; it has no free-running capability. In our generalization, we replace the state set  $Q$  by a CTS  $C$ , the free monoid  $X$  by an arbitrary trace algebra, and we let  $\delta$  be a monoid homomorphism from  $X$  to the monoid of “orthogonal” endomorphisms of  $C$ ,

defined formally below. The CTS  $C$  describes the free-running capabilities of the machine, and  $\delta$  describes the effect of input. We emphasize that  $\delta(x)$  acts on transitions of  $C$ , as well as on states.

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

- (1) for all transitions  $t$  of  $C$ , if  $\mu(t) = \text{id}$ , then  $t = \text{id}$ ;
- (2) for all coinital 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  $\delta : X \rightarrow \text{CTS}(C, C)$ , such that for each  $x \in X$ , the morphism  $\delta(x)$  is orthogonal. We usually write  $\delta_x$ , instead of  $\delta(x)$ , for the application of an action  $\delta$  to argument  $x \in X$ .

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

- $C$  is a CTS, called the *underlying CTS* of  $\mathcal{M}$ ,
- $q_*$  is a distinguished state of  $C$ , called the *start state* of  $\mathcal{M}$ ,
- $\lambda : C \rightarrow Y$  is a morphism, called the *output map* of  $\mathcal{M}$ ,
- $\delta$  is an action of  $X$  on  $C$ , called the *input map* of  $\mathcal{M}$ ,

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

Machine  $M$  is called a *Kahn machine* if its underlying CTS  $C$  is determinate. In the special case that  $X = 1$  (the terminal object in 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  $q_*$ , and its output map  $\lambda : C \rightarrow Y$ , since there is only one possible input map  $\delta : 1 \rightarrow \text{CTS}(C, C)$ .

We require the orthogonality property in the definition of a machine so that the definition of the “feedback” operation on machines (see Section 3.2.4) and the related construction of an automaton from a machine (see Section 4.1) make sense.

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

- (1)  $q'_* = \mu(q_*)$ ,
- (2)  $\lambda = \lambda' \circ \mu$ ,
- (3)  $\delta'_x \circ \mu = \mu \circ \delta_x$  for all  $x \in X$ .

Let  $\text{Mach}_{X,Y}$  denote the category of  $(X, Y)$ -machines and their morphisms, and let  $\text{Auto}_Y = \text{Mach}_{1,Y}$  be 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 *monotone sequential dataflow process (MSDP)* with  $m$  *input ports* and  $n$  *output ports* is a triple  $(Q, q_*, A)$ , where

- $Q$  is a set of *states*, with  $q_* \in Q$  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  $((q, \varepsilon), \varepsilon, (q, \varepsilon))$ ;

(2) if the set  $A$  contains a transition  $((q, x'), y, (q', x''))$ , then  $A$  also contains a transition  $((q, x'x), y, (q', x''x))$  for each  $x \in X$ .

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. This is the reason for using the adjective “monotone” to describe these processes.

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 MSDP. 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 MSDP. An example of a nonfunctional process also representable as an MSDP 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.

An MSDP  $(Q, q_i, A)$  may be regarded as an  $(X, Y)$ -machine  $\mathcal{M} = (C, q_i, \lambda, \delta)$  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 all pairs  $(q, x)$  with  $q \in Q$  and  $x \in X$ , and as proper transitions the elements of  $A$ . Define

$$\text{dom}((q, x), y, (q', x')) = (q, x),$$

$$\text{cod}((q, x), y, (q', x')) = (q', x').$$

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  $\lambda: C \rightarrow Y$  take  $((q, x), y, (q', x'))$  to  $y$ .
- Let  $\delta: X \rightarrow \text{CTS}(C, C)$  be defined so that  $\delta_x$  takes a transition  $((q, x'), y, (q', x'')) \in A$  to the transition  $((q, x'x), y, (q', x''x)) \in A$ , which exists by condition (2) in the definition of an MSDP above.

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

### 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 Fig. 4. The “adjointness” results of Lemmas 3.3.1 and 3.3.2 provide motivation for considering this particular collection of operations.

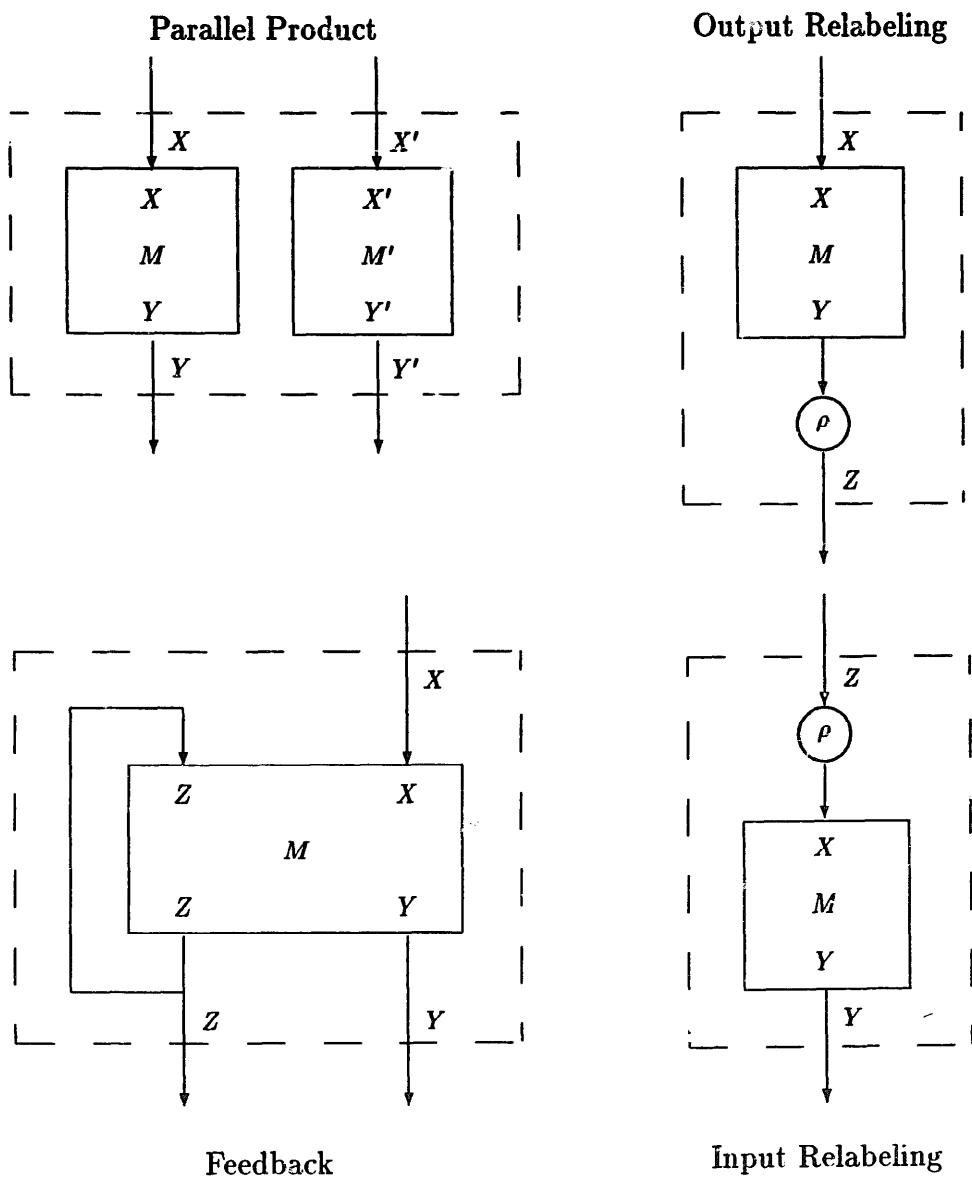


Fig. 4. Operations on machines.

### 3.2.1. Parallel product

Suppose  $\mathcal{M} = (C, q_i, \lambda, \delta)$  is an  $(X, Y)$ -machine and  $\mathcal{M}' = (C', q'_i, \lambda', \delta')$  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

$$\mathcal{M} \times \mathcal{M}' = (C \times C', (q_i, q'_i), \lambda' \times \lambda, \delta''),$$

where  $\delta''_{x:x'} = \delta_x \times \delta'_{x'}$  for each  $x:x' \in X \times X'$ . Parallel product is easily seen to be the object map of a functor from  $\text{Mach}_{X,Y} \times \text{Mach}_{X',Y'}$  to  $\text{Mach}_{X \times X', Y \times Y'}$ .

### 3.2.2. Output relabeling

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

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

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

### 3.2.3. Input relabeling

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

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

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

### 3.2.4. Feedback

Given a  $(Z \times X, Y \times Z)$ -machine  $\mathcal{M}$ , we wish to define an  $(X, Y \times Z)$ -machine  $\{\mathcal{M}\}_{\odot Z}$ , which represents the machine  $\mathcal{M}$  with its  $Z$ -output “fed back” to its  $Z$ -input. Each transition  $t: q \rightarrow r$  in the underlying CTS of  $\mathcal{M}$  generates some feedback input  $\lambda_Z(t)$ . To account for this feedback input, transition  $t$  in  $\{\mathcal{M}\}_{\odot Z}$  will not have codomain  $r$ , but rather it will have codomain  $\delta_{\lambda_Z(t): \varepsilon_X}(r)$ . Thus, each transition  $t$  of  $\{\mathcal{M}\}_{\odot Z}$  can be thought of as a transition of  $\mathcal{M}$  that has been “composed” with the effect, given by  $\delta_{\lambda_Z(t): \varepsilon_X}$ , of the feedback input generated by  $t$ .

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

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

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:

$$\text{dom}'(t) = \text{dom}(t), \quad \text{cod}'(t) = \delta_{\lambda_Z(t): \varepsilon_X}(\text{cod}(t)).$$

The identities of  $C'$  are the same as those of  $C$ . For coinitial  $t, u$ , let  $t$  and  $u$  be consistent in  $C'$  iff they are consistent in  $C$ , and define

$$t \uparrow' u = \delta_{\lambda_Z(u): \varepsilon_X}(t \uparrow u).$$

- $\lambda': C' \rightarrow Y$  is defined by  $\lambda'(t) = \lambda(t)$ .

- $\delta': X \rightarrow \text{CTS}(C', C')$  is defined by  $\delta'_x(t) = \delta_{\epsilon_{Z:x}}(t)$ .

It is straightforward to verify that  $\{\mathcal{M}\}_{\bigcirc Z}$  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).

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

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

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  $\mathcal{M}$ .

We next associate with each  $Y$ -automaton  $\mathcal{A}$  an *output set*, which represents the set of all possible outputs that can be produced in the various runs of  $\mathcal{A}$ . Output sets of automata are special cases of *input/output relations* of machines, which we will define and investigate in more detail in Section 4. Formally, suppose  $\mathcal{A} = (C, q_*, \lambda)$  is a  $Y$ -automaton. The *computations* of  $\mathcal{A}$  are the computations of the pointed CTS  $(C, q_*)$ . A computation  $J$  of  $\mathcal{A}$  is *maximal* if it is not properly included in any other computation of  $\mathcal{A}$ . Let  $\eta_C : (q_* \downarrow C) \rightarrow (C, q_*)$  denote the “natural map” associated with the computation diagram construction (a right adjoint). Define the *synchronization diagram*  $\text{Diag}(\mathcal{A})$  of  $\mathcal{A}$  to be the pair  $((q_* \downarrow C), \lambda^\dagger)$ , where  $\lambda^\dagger = \lambda \circ \eta_C$ . Intuitively,  $\text{Diag}(\mathcal{A})$  is obtained by “unwinding”  $\mathcal{A}$  to obtain its computation diagram, and labeling transitions in this diagram consistently with the corresponding transitions in  $\mathcal{A}$ . The induced map  $(\lambda^\dagger)^*: (q_* \downarrow C^*) \rightarrow Y$  labels each event of  $\mathcal{A}$  by its output trace. This map extends uniquely to a continuous map  $\bar{\lambda}$  from the cpo of computations of  $\mathcal{A}$  to the ideal space  $\bar{Y}$  of  $Y$ . Thus, each computation  $J$  of  $\mathcal{A}$  determines an ideal  $\bar{\lambda}(J) \in \bar{Y}$ , which we call the *complete output trace* of  $J$ . The *output set*  $\text{Out}(\mathcal{A})$  of  $\mathcal{A}$  is the set of all complete output traces determined by maximal computations of  $\mathcal{A}$ .

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

$$\text{Out}(\mathcal{M}, \mathcal{N}) = \text{Out}(\mathcal{M}', \mathcal{N})$$

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 [11].

We wish to show that observable equivalence is a congruence with respect to the operations on machines. This can be done with the help of some lemmas, which are intended to be suggestive of results on adjoint operators in linear algebra.

**Lemma 3.3.1.**

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

$$\langle \mathcal{M}, \mathcal{N} \rangle = \langle \mathcal{N}, \mathcal{M} \rangle \triangleright \pi.$$

- (2) Suppose  $\mathcal{M}$  is an  $(X, Y)$ -machine,  $\mathcal{N}$  is a  $(Z, X)$ -machine, and  $\rho: Y \rightarrow Z$ . Then

$$\langle \mathcal{M} \triangleright \rho, \mathcal{N} \rangle = \langle \mathcal{M}, \rho \triangleright \mathcal{N} \rangle \triangleright (\text{id}_X \times \rho).$$

- (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

$$\langle \mathcal{M} \times \mathcal{N}, \mathcal{P} \rangle = \langle \mathcal{M}, \{\mathcal{N} \times \mathcal{P}\}_{\circ Z \times w} \rangle.$$

**Proof.** Straightforward.  $\square$

A result exactly analogous to (2) and (3) does not hold with  $\langle \{\mathcal{M}\}_{\circ Z}, \mathcal{N} \rangle$  on the left-hand side of the isomorphism. Intuitively, the reason is that the feedback operation we have defined causes the application of feedback input to occur simultaneously with the transition that generates it. For  $\langle \{\mathcal{M}\}_{\circ Z}, \mathcal{N} \rangle$  to be isomorphic to  $\langle \mathcal{M}, \mathcal{N} \rangle$  we would need a machine  $\mathcal{N}'$  that could transfer input instantaneously from its input to its output, a capability that our machines do not have. However, by using as a “buffer” a  $(Z, Z)$ -machine  $\mathcal{I}_Z$  that simply passes its input through unchanged to its output, we can obtain a somewhat weaker result (Lemma 3.3.2 below), which is adequate for our purposes. Lemma 4.6.2 provides the machine  $\mathcal{I}_Z$ , 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  $\rho: Y \times Z \rightarrow Y \times Z \times Z$  be the morphism that takes  $y:z$  to  $y:z:z$ , and let  $\pi: Z \times X \times Y \times Z \rightarrow X \times Y \times Z$  be the morphism that takes  $z':x:y:z$  to  $x:y:z$ . Then

$$\text{Out}(\{\mathcal{M}\}_{\circ Z}, \mathcal{N}) = \text{Out}(\langle \mathcal{M}, \rho \triangleright (\mathcal{N} \times \mathcal{I}_Z) \rangle \triangleright \pi).$$

**Proof.** Suppose  $\mathcal{A} = \langle \{\mathcal{M}\}_{\circ Z}, \mathcal{N} \rangle = (C, q_i, \lambda)$  and

$$\mathcal{A}' = \langle \mathcal{M}, \rho \triangleright (\mathcal{N} \times \mathcal{I}_Z) \rangle \triangleright \pi = (C', q'_i, \lambda').$$

We can construct CCDiag morphisms

$$\rho: (q_i \downarrow C^*) \rightarrow (q'_i \downarrow (C')^*) \quad \text{and} \quad \rho': (q'_i \downarrow (C')^*) \rightarrow (q_i \downarrow C^*),$$

such that  $(\lambda')^\dagger = \lambda^\dagger \circ \rho$ ,  $\rho' \circ \rho = \text{id}$ , and  $t' \leq \rho(\rho'(t'))$  for all events  $t'$  in  $(q'_i \downarrow (C')^*)$ . From this, it follows that the output sets of  $\mathcal{A}$  and  $\mathcal{A}'$  are identical. We omit the details.  $\square$

**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)  $\mathcal{M} \triangleright \rho \sim \mathcal{M}' \triangleright \rho$  for all morphisms  $\rho : Y \rightarrow Z$ ;
  - (c)  $\rho \triangleright \mathcal{M} \sim \rho \triangleright \mathcal{M}'$  for all morphisms  $\rho : Z \rightarrow X$  for which there exists a morphism  $\rho' : X \rightarrow Z$  with  $\rho' \circ \rho = \text{id}_Z$ .
- (2) *Suppose  $\mathcal{M}$  and  $\mathcal{M}'$  are  $(Z \times X, Y \times Z)$ -machines. If  $\mathcal{M} \sim \mathcal{M}'$ , then  $\{\mathcal{M}\}_{\circ Z} \sim \{\mathcal{M}'\}_{\circ Z}$ .*

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

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

$$\begin{aligned} \text{Out}(\mathcal{M} \times \mathcal{N}, \mathcal{P}) &= \text{Out}(\mathcal{M}, \{\mathcal{N} \times \mathcal{P}\}_{\circ Z \times W}) \\ &= \text{Out}(\mathcal{M}', \{\mathcal{N} \times \mathcal{P}\}_{\circ Z \times W}) \\ &= \text{Out}(\mathcal{M}' \times \mathcal{N}, \mathcal{P}). \end{aligned}$$

(1)(b): Similar to (1)(a), but using Lemma 3.3.1(2).

(1)(c): Similar to (1)(b), but using the isomorphism

$$\langle \rho \triangleright \mathcal{M}, \mathcal{N} \rangle = \langle \mathcal{M}, \mathcal{N} \triangleright \rho \rangle \triangleright (\rho' \times \text{id}_Y),$$

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

(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 such a congruence on machines, which 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

$$\text{Out}(\mathcal{M}, \mathcal{N}) \neq \text{Out}(\mathcal{M}', \mathcal{N}).$$

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.  $\square$

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 [21] and more conclusively by Brock and Ackerman [8] (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 [3, 23, 24, 37, 38, 40] have proposed process models that incorporate somewhat more information than just input/output relations. Abramsky [1] has shown the full abstractness of a power domain 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. Recently, a full abstraction result has been claimed by Kok [25].

#### 4. An analysis of feedback

It would seem that a deep understanding of the feedback operation is prerequisite to a satisfactory resolution of 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  $\{\mathcal{M}\}_{\square_Z}$ .

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, 10]. Our second mapping “unwinds” automata to obtain their synchronization diagrams,” which are a generalization of “synchronization trees” [47]. 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 [38, 39]. 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, 8]. 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 algebra of all machines. The mapping from scenarios 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  $\bar{X}$  to  $\bar{Y}$ , 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 [19], and has been called the “Kahn Principle.” Although the Kahn Principle has been proved before [13], our proof applies to a more general, axiomatically defined class of processes.

#### 4.1. Automata

Given  $\bar{x} \in \bar{X}$ , let  $\hat{X}$  denote the  $(Y, X)$ -machine  $(X, q_c, \text{id}_X, \delta)$ , where  $q_c$  is the unique proper state of  $X$  and  $\delta_y : X \rightarrow X$  is  $\text{id}_X$  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  $M$  determines a corresponding  $(X \times Y)$ -automaton  $\text{Auto}(M)$ , under the definition

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

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

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  $\lambda : C \rightarrow X$  is a morphism. We say that a proper transition  $t$  of  $\mathcal{A}$  is *canonical* w.r.t.  $\lambda$ , if for any other proper transition  $t'$  of  $C$ , with  $\text{dom}(t) = \text{dom}(t')$  and  $\lambda(t') = \lambda(t)$ , we have  $t \leq t'$ . Note that canonical transitions, when they exist, are uniquely determined by their domain and their image under  $\lambda$ .

**Theorem 4.1.** An  $(X \times Y)$ -automaton  $\mathcal{A} = (C, q_c, \lambda)$  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  $\text{dom}(x_q) = q$  and  $\lambda_X(x_q) = x$ , such that  $x_q$  is canonical w.r.t.  $\lambda_X$ .
- (2) Each proper transition  $t : q \rightarrow r$  of  $\mathcal{A}$ , has a decomposition  $t = x_q \vee u$ , where  $x = \lambda_X(t)$ .
- (3) For each proper transition  $t : q \rightarrow r$  of  $\mathcal{A}$ , and each  $x \in X$ , we have  $x_q \uparrow t = (x \uparrow \lambda(t))_r$ .
- (4) For each proper transition  $t : q \rightarrow r$  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  $t: q \rightarrow r$  of  $\mathcal{A}$ , and each  $x \in X$ , if  $\lambda_X(t) = \varepsilon$  and  $t \uparrow x_q = \text{id}_r$ , then  $t = \text{id}_q$ .

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  $\mathcal{A} = \text{Auto}(\mathcal{M})$  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  $\mathcal{A} = \text{Auto}(\mathcal{M})$  for some  $\mathcal{M}$ , and if  $\mathcal{A}$  has property (6), then  $\mathcal{M}$  can be chosen to be a Kahn machine.

( $\Rightarrow$ ): Suppose  $\mathcal{A} = \text{Auto}(\mathcal{M})$ . Since properties (1)–(5) are preserved under isomorphism, we may suppose without loss of generality that  $\mathcal{A} = \text{Auto}(\mathcal{M})$ . 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  $x_q = (\text{id}_q, x)$  are the canonical transitions required by property (1). The remaining properties then follow by straightforward calculations, which we omit.

( $\Leftarrow$ ): Suppose  $\mathcal{A} = (C, q_*, \lambda)$  has properties (1)–(5). We show how to construct  $\mathcal{M}$  so that  $\mathcal{A} = \text{Auto}(\mathcal{M})$ . It is easily verified that the states of  $C$ , equipped with all transitions  $t$  of  $C$  such that  $\lambda_X(t) = \varepsilon$ , define a sub-automaton  $\mathcal{A}' = (C', q_*, \lambda')$  of  $\mathcal{A}$ . Let  $\delta: X \rightarrow \text{CTS}(C', C')$  be defined by  $\delta_x(t) = t \uparrow x_{\text{dom}(q)}$ . Straightforward calculations, which we omit, from the definitions and properties (1)–(5) suffice to verify that  $\mathcal{M} = (C', q_*, \lambda'_Y, \delta)$  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  $\mathcal{A} = \text{Auto}(\mathcal{M})$ . The required isomorphism  $\mu: \text{Auto}(\mathcal{M}) \rightarrow \mathcal{A}$  is obtained as the morphism that takes each transition  $(u, x): \text{dom}(u) \rightarrow \delta_x(\text{cod}(u))$  of  $\text{Auto}(\mathcal{M})$  to the join  $x_q \vee u$  in  $\mathcal{A}$ , which exists by property (4). The morphism  $\mu$  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  $x_q \uparrow u' = x_{\text{cod}(u')}$  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  $u = u'$ .  $\square$

In case  $\mathcal{A} = (C, q_*, \lambda)$  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  $\lambda^{\text{in}}$  and  $\lambda^{\text{out}}$  as alternate notations for  $\lambda_X$  and  $\lambda_Y$ , respectively.

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

$$(-) \triangleright (\pi_Z: \text{id}_{W \times Z}): \text{Auto}_{W \times Z} \rightarrow \text{Auto}_{Z \times W \times Z}$$

has a right adjoint

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

whose object map takes each  $(Z \times W \times Z)$ -automaton  $\mathcal{A} = (C, q_0, \lambda)$  to the  $(W \times Z)$ -automaton  $\{\mathcal{A}\}_{=Z} = (C', q_0, \lambda')$ , 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  $\lambda'$  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

$$\mathbf{Auto}(\{\mathcal{M}\}_{\circ Z}) \simeq \{\mathbf{Auto}(\mathcal{M})\}_{=Z}.$$

**Proof.** If  $\mathcal{M} = (C, \iota, \lambda, \delta)$ , then the required isomorphism maps a transition  $(t, x)$  of  $\mathbf{Auto}(\{\mathcal{M}\}_{\circ Z})$  to the corresponding transition  $(t, \lambda_Z(t):x)$  of  $\{\mathbf{Auto}(\mathcal{M})\}_{=Z}$ . The details are straightforward, and are omitted.  $\square$

#### 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. Also, we will not bother to distinguish notationally between  $\lambda$  and the mapping it induces from events of  $D$  to  $W$ . Let  $\mathbf{Diag}_W$  denote the full subcategory of  $\mathbf{Auto}_W$ , whose objects are the  $W$ -synchronization diagrams. The map  $\mathbf{Diag} : \mathbf{Auto}_W \rightarrow \mathbf{Diag}_W$  defined in Section 3.3 is easily seen to be the object map of a functor, right-adjoint to the inclusion of  $\mathbf{Diag}_W$  in  $\mathbf{Auto}_W$ .

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

**Lemma 4.2.1.** An  $(X \times Y)$ -synchronization diagram is an  $(X, Y)$ -input/output diagram (respectively  $(X, Y)$ -Kahn diagram) iff it is an  $(X, Y)$ -input/output automaton (respectively  $(X, Y)$ -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  $\mathbf{Diag}(\mathcal{A})$  is obtained from  $\mathcal{A}$ .  $\square$

**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 event  $t_x$  of  $D$  such that  $\lambda_X(t_x) = x$ , and such that if  $u$  is any event of  $D$  with  $\lambda_X(u) = \lambda_X(t_x)$ , then  $t_x \leq u$ .
- (2) If  $u$  is an event 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 events 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 and Theorem 4.1.  $\square$

The events  $t_x$  in (1) above are called the *pure-input events* 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  $\lambda_{Z_1}$  and  $\lambda_{Z_2}$  be the projections of  $\lambda$  on the first and second  $Z$  components, respectively. Define a *feedback computation sequence* for  $\mathcal{D}$  to be a computation sequence  $t_0 \leq t_1 \leq \dots \leq t_n$  for  $\mathcal{D}$ , such that  $\lambda_{Z_1}(t_k) = \lambda_{Z_2}(t_k)$  for  $1 \leq k \leq n$ . An event 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 event 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  $\{\mathcal{D}\}_{=Z} = (D', \lambda')$ , where  $D'$  is the feedback-reachable subdiagram of  $D$ , and  $\lambda'$  is the restriction to  $D'$  of  $\lambda_{W \times Z}$ .

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

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

**Proof.** Straightforward from the observation that both functors

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

and

$$\{\text{Diag}(-)\}_{=Z} : \text{Auto}_{Z \times W \times Z} \rightarrow \text{Diag}_{W \times Z},$$

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

$$- \triangleright (\pi_Z : \text{id}_{W \times Z}) : \text{Auto}_{W \times Z} \rightarrow \text{Auto}_{Z \times W \times Z}$$

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

#### 4.3. Behaviors

If  $\mathcal{D} = (D, \lambda)$  is a  $W$ -synchronization diagram, then each consistent set  $J$  of events of  $\mathcal{D}$  determines a consistent subset  $\lambda(J)$  of  $W$ , which we call the *history* of  $J$  with respect to  $\lambda$ . The set  $\lambda(J)$  extends to a least ideal  $\bar{\lambda}(J) \in \bar{W}$ , which we call the *complete trace* of  $J$  with respect to  $\lambda$ . If  $\mathcal{D}$  is an  $(X, Y)$ -input/output diagram, and  $\bar{\lambda}(J) = \bar{x} : \bar{y}$ , then we call  $\bar{x}$  the *complete input trace*, and  $\bar{y}$  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 events of  $D$  extends to a behavior of  $\mathcal{D}$  having the same complete input trace as  $J$ . In particular, for each  $\bar{x} \in \bar{X}$  there is a behavior of  $\mathcal{D}$  with  $\bar{x}$  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  $\bar{x}$ , and  $\bar{x} \leq \bar{x}'$ , then  $J$  extends to a behavior  $J'$  of  $\mathcal{D}$  with complete input trace  $\bar{x}'$ .
- (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 events of  $D$ . Then the class of all consistent sets of events 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 events of  $D$  whose input traces are prefixes of  $\bar{x}$ , this construction yields a behavior of  $\mathcal{D}$  with complete input trace  $\bar{x}$ . Moreover, the existence of two distinct behaviors of  $\mathcal{D}$  with the same complete input trace implies the existence of two inconsistent events 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  $\bar{x}$ , and given  $\bar{x}'$  with  $\bar{x} \leq \bar{x}'$ , let  $K$  be the set of all pure-input events of  $D$  whose input traces are prefixes of  $\bar{x}'$ . Then the set  $J \cup K$  is consistent by Lemma 4.2.2, and has complete input trace  $\bar{x}'$ , hence it extends to a behavior  $J'$  of  $\mathcal{D}$  that has complete input trace  $\bar{x}'$ .

(3): Suppose the set  $J = \bigvee\{J_i : i \in I\}$  were not a behavior of  $\mathcal{D}$ . Then there would exist some event  $t$  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.  $\square$

If  $(K, \leq)$  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  $t \leq u$ .

**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  $J$  and  $K$  have complete trace  $\bar{z}:\bar{w}:\bar{z}$ . Let  $t \in K$  be given; then we can choose  $u \in J$  such that  $\lambda(t) \leq \lambda(u)$ . Since  $K$  is a computation,  $t$  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  $t \leq t \vee u$ , as required.  $\square$

**Theorem 4.4.** Suppose  $\mathcal{D} = (D, \lambda)$  is a  $(Z \times X, Y \times Z)$ -input/output diagram. Then a set  $J$  of events of  $D$  is a behavior of  $\{\mathcal{D}\}_{=Z}$  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  $\{\mathcal{D}\}_{=Z}$ , with  $\bar{\lambda}(J) = \bar{z}:\bar{x}:\bar{y}:\bar{z}$ . By Lemma 4.3.1,  $J$  extends to a behavior  $K$  of  $\mathcal{D}$ , with complete input trace  $\bar{z}:\bar{x}$ . 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  $J$ , hence is the result of a feedback computation sequence in  $K$ , because  $J \subseteq K$ . 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  $\bar{x}$ , hence is in  $J$  because  $J$  is a behavior of  $\{\mathcal{D}\}_{=Z}$ . Thus  $J = J'$ .

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

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  $t_0 \leq t_1 \leq \dots \leq t_{n+1}$  of  $K$ . Applying the induction hypothesis to the computation sequence  $t_0 \leq t_1 \leq \dots \leq t_n$ , we obtain an element  $u_n$  of  $J$  with  $t_n \leq u_n$ . Let  $v = t_{n+1} \uparrow t_n$ , then  $v$  is an atom by definition of a computation sequence. (See Fig. 5.) Now,  $\lambda_Z^{\text{in}}(t_{n+1}) \leq \bar{z}$ , because  $t_{n+1} \in K$  and  $\lambda_Z^{\text{in}}(K) = \bar{z}$ . Since we also have  $\bar{\lambda}_Z^{\text{in}}(J) = \bar{z}$ , and  $u_n$  is an element of  $J$ , there must exist an element  $u'_n$  of  $J$ , with  $u_n \leq u'_n$  and  $\lambda_Z^{\text{in}}(t_{n+1}) \leq \lambda_Z^{\text{in}}(u'_n)$ . Let  $v' = v \uparrow (u'_n \uparrow t_n)$ ; then  $\lambda_Z^{\text{in}}(v') = \varepsilon$ . Moreover,  $v'$  is an atom because  $v$  is. Let  $v''$  be the pure-input transition of  $D$  with  $\text{dom}(v'') = \text{dom}(v')$  and  $\lambda^{\text{in}}(v'') = \lambda_Z^{\text{out}}(v'): \varepsilon_X$ . 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  $t_{n+1} \leq u_{n+1}$ .

To complete the induction step, it remains to be shown that  $u_{n+1} \in J$ . It suffices, since  $J$  is a behavior of  $\{\mathcal{D}\}_{=Z}$ , to show that  $\lambda_X(u_{n+1}) \leq \bar{x}$  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,  $\lambda_X(u_{n+1}) \leq \bar{x}$  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  $\bar{z}:\bar{x}$ . We claim that  $J$  is a behavior of  $\{\mathcal{D}\}_{=Z}$ ; that is,  $J$  is maximal among all computations  $J'$  of  $\{\mathcal{D}\}_{=Z}$  with  $\bar{\lambda}_X(J') = \bar{x}$ . To show this, we show that if  $t$  is a feedback-reachable event of  $D$ , consistent with  $J$ , and such that  $\lambda_X(t) \leq \bar{x}$ , then  $t \in K$ . It then follows that  $t \in J$  because  $J$  is the set of feedback-reachable elements of  $K$ .

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

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

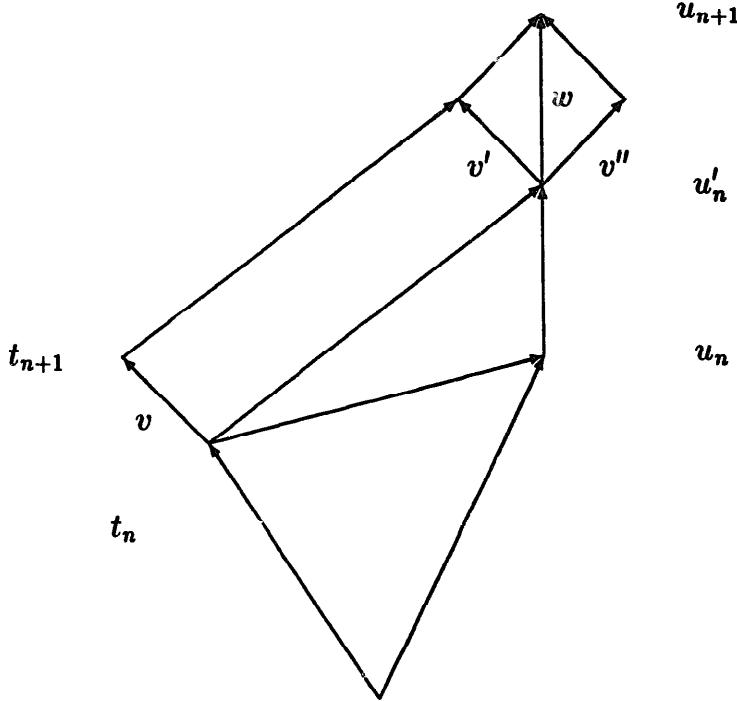


Fig. 5. Proof of Theorem 4.4.

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  $t_0 \leq t_1 \leq \dots \leq t_{n+1}$ , 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  $\lambda_Z^{\text{in}}(t_{n+1}) = \lambda_Z^{\text{out}}(t_{n+1})$ , 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  $\lambda^{\text{in}}(v) = \lambda_Z^{\text{in}}(u) : \epsilon_X$ . Then  $t_n w$  is an event of  $D$  that is consistent with  $J$ , and has the property  $\lambda_Z^{\text{in}}(t_n w) = \lambda_Z^{\text{in}}(t_n) \leq \bar{z}$ . 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,

$$\lambda_Z^{\text{in}}(t_{n+1}) = \lambda_Z^{\text{out}}(t_{n+1}) = \lambda_Z^{\text{out}}(t_n u) = \lambda_Z^{\text{out}}(t_n w) \leq \bar{z}.$$

But then  $t_{n+1}$  must be in  $K$ , since it is consistent with  $J$ , hence with  $K$ , we have  $\lambda_Z^{\text{in}}(t_{n+1}) \leq \bar{z}$ , and  $K$  is a behavior.  $\square$

#### 4.4. Histories

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

$$\epsilon_X : \epsilon_Y = x_0 : y_0 \leq x_1 : y_1 \leq \dots \leq x_n : y_n = x : y,$$

such that for each  $k$  with  $0 \leq k \leq n$ , the trace  $x_k : y_{k+1}$  is in  $H$ . We call such a sequence a *computation sequence* for  $H$ , and the  $x:y$  its *result*. If  $H$  is an  $(X, Y)$ -history, then

the ideal  $\bar{x}:\bar{y} = \bigvee H \in \bar{X} \times \bar{Y}$  is called the *complete trace* of  $H$ , with  $\bar{x}$  called the *complete input trace* and  $\bar{y}$  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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n,$$

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  $\{H\}_{\circ Z}$ .

**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  $\{H\}_{\circ Z}$  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  $G = \{H\}_{\circ Z}$ .

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  $t_0 \leq t_1 \leq \dots \leq t_n$  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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n$$

is a feedback computation sequence for  $H$ , thus showing that  $x:y:z = x_n:y_n:z_n \in \{H\}_{\circ Z}$ . 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  $\text{dom}(v_k) = \text{dom}(u_k)$  and trace  $\lambda^{\text{in}}(u_k):\varepsilon_{Y \times Z}$ . Then  $w_k$  has trace  $\varepsilon_{Z \times X}:\lambda^{\text{out}}(u_k)$ , so  $t_k w_k$  has trace  $z_k:x_k:y_{k+1}:z_{k+1}$ . Since  $t_k w_k \leq t_{k+1} \in K$ , 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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n$$

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

$$t_0 \leq t_1 \leq \dots \leq t_{m_n}$$

for  $K$ , and nonnegative integers  $0 = m_0, m_1, \dots, m_n$ , 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 a computation sequence  $t_0 \leq t_1 \leq \dots \leq t_{m_k} \in K$ , 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  $t_{m_k} \leq u_k$ . Thus, we may obtain a computation sequence  $v_0 \leq v_1 \leq \dots \leq v_p \in K$ , with  $v_0 = t_{m_k}$  and  $v_p = u_k$ . For  $0 \leq i \leq p$ , let  $w_i$  be the pure-input event of  $D$  with trace  $\lambda_Z^{\text{out}}(v_i):\varepsilon_X:\varepsilon_Y:\varepsilon_Z$ . Note that then  $v_i \vee w_i$  exists for each  $i$  with  $0 \leq i \leq p$ , and we have that  $(v_{i+1} \vee w_{i+1}) \uparrow (v_i \vee w_i)$  is an atom 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  $\lambda^{\text{in}}(w_i) \leq z_{k+1}$ , 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

$$t_0 \leq t_1 \leq \dots \leq t_{m_k} \leq t_{m_k+1} \leq \dots \leq t_{m_{k+1}}$$

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

**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  $\bar{x}:\bar{y}:\bar{z}$ , is a history of  $\{\mathcal{D}\}_{=Z}$  iff  $G = \{H\}_{\circ Z}$  for some history  $H$  of  $\mathcal{D}$  with complete trace  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$ .

**Proof.** If  $G$  is a history of  $\{\mathcal{D}\}_{=Z}$ , with complete trace  $\bar{x}:\bar{y}:\bar{z}$ , then  $G$  is the history of some behavior  $J$  of  $\{\mathcal{D}\}_{=Z}$ . 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  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$  by Lemma 4.3.2. Moreover,  $G = \{H\}_{\circ Z}$  by Lemma 4.4.1.

Conversely, suppose  $G = \{H\}_{\circ Z}$  for some history  $H$  of  $\mathcal{D}$ . Suppose  $G$  has complete trace  $\bar{x}:\bar{y}:\bar{z}$  and  $H$  has complete trace  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$ . 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-reachable elements of  $K$ , from which it follows by Lemma 4.3.2 that  $J$  is cofinal in  $K$ . Thus,  $J$  is a behavior of  $\{\mathcal{D}\}_{=Z}$ , and  $G$  is a history of  $\{\mathcal{D}\}_{=Z}$ .  $\square$

#### 4.5. Scenarios

An  $(X, Y)$ -Kahn function is a continuous function  $\phi$  from an initial segment (a nonempty, downward-closed, and directed-complete subset) of  $\text{dom}(\phi)$  of  $\bar{X}$  to  $\bar{Y}$ . If  $\text{dom}(\phi) = \bar{X}$ , then  $\phi$  is called total. If  $U$  is an initial segment of  $X$ , 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  $\phi$  is a  $(Z \times X, Y \times Z)$ -Kahn function. We say that  $\phi$  is feedback-compatible if  $\bar{y}:\bar{z} \in \phi(\text{dom}(\phi))$  implies  $\bar{z}:\bar{x} \in \text{dom}(\phi)$  for all  $\bar{x} \in \pi_{\bar{X}}(\text{dom}(\phi))$ . Note that total  $(Z \times X, Y \times Z)$ -Kahn functions are always feedback-compatible. If  $\phi$  is feedback-compatible, then define the feedback functional  $\Phi$  associated with  $\phi$ , to be the functional

$$\Phi : [\pi_{\bar{X}}(\text{dom}(\phi)) \rightarrow \bar{Y} \times \bar{Z}] \rightarrow [\pi_{\bar{X}}(\text{dom}(\phi)) \rightarrow \bar{Y} \times \bar{Z}]$$

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

$$\Phi(\psi) = \phi \circ ((\pi_{\bar{Z}} \circ \psi) : \text{id}_{\text{dom}(\psi)}).$$

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

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

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

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

**Proof.** Standard.  $\square$

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

$$(\vee \text{dom}(\phi)) : (\vee \phi(\text{dom}(\phi)))$$

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

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

$$\phi : \{\bar{x} \in \bar{X} : \bar{x} \leq \bar{x}_0\} \rightarrow \bar{Y},$$

according to the definition

$$\phi(\bar{x}) = \vee \{y \in Y : \exists x : y \in H, x \leq \bar{x}\}.$$

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

Brock and Ackerman [8, 7] 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  $\bar{x}_0 : \bar{y}_0$ ,  $\bar{x}$  “enables”  $\bar{y}$  iff every finite prefix  $x$  of  $\bar{x}_0$  that “must precede” some finite prefix  $y$  of  $\bar{y}$  is already a prefix of  $\bar{x}$ , and  $x$  “must precede”  $y$  iff for all  $\bar{x} \leq \bar{x}_0$ , if  $\bar{x}$  “enables”  $y$ , then  $x \leq \bar{x}$ . 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  $\phi$ . Let  $G = \{H\}_{\circ Z}$ , and let  $\psi$  be the scenario of  $G$ . If  $\phi$  is feedback-compatible, then  $\psi = \{\phi\}_{\circ Z}$ .

**Proof.** We show  $(\Rightarrow)$   $\psi(\bar{x}) \leq \{\phi\}_{\circ Z}(\bar{x})$  for all  $\bar{x} \in \bar{X}$ , and  $(\Leftarrow)$   $\psi$  is a fixed point of the feedback functional associated with  $\phi$ .

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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n$$

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

$$y_{k+1}:z_{k+1} \leq \phi^{(k+1)}(\bar{x}),$$

where  $\phi^{(k)}$  is as defined in Lemma 4.5. It follows that  $y:z \leq \{\phi\}_{\circ Z}(\bar{x})$ . Since  $y:z$  was an arbitrary finite prefix of  $\bar{y}:\bar{z}$ , it follows that  $\bar{y}:\bar{z} \leq \{\phi\}_{\circ Z}(\bar{x})$ , as was to be shown.

$(\Leftarrow)$ : Since we already know that  $\psi \sqsubseteq \{\phi\}_{\circ Z}$ , to show that  $\psi$  is a fixed point of the feedback functional  $\Phi$  associated with  $\phi$ , it remains only to show that  $\Phi(\psi) \sqsubseteq \psi$ . That is, we must show that  $\phi(\pi_Z(\psi(\bar{x})):\bar{x}) \leq \psi(\bar{x})$  for  $\bar{x} \in \bar{X}$ . 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  $y' \leq y''$  and  $z' \leq z''$ .

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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n$$

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

$$z_0:x_0:y_0:z_0 \leq z_1:x_1:y_1:z_1 \leq \dots \leq z_n:x_n:y_n:z_n \leq z'':x:y'':z'',$$

is a feedback computation sequence for  $H$ . It follows that  $x:y'':z''$  is in  $G$ .  $\square$

**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  $\psi$ , with complete trace  $\bar{x}:\bar{y}:\bar{z}$ , is a scenario of  $\{\mathcal{D}\}_{=Z}$  iff  $\psi = \{\phi\}_{\circ Z}$ , where  $\phi$  is a scenario of  $\mathcal{D}$  with complete trace  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$ .

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

Conversely, if  $\psi = \{\phi\}_{\circ Z}$  has complete trace  $\bar{x}:\bar{y}:\bar{z}$ , where  $\phi$  is a scenario of  $\mathcal{D}$  with complete trace  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$ , then  $\phi$  is the scenario of some history  $H$  of  $\mathcal{D}$ . By Theorem 4.5,  $\{H\}_{\circ Z}$  is a history of  $\{\mathcal{D}\}_{=Z}$ , and by Lemma 4.5.2,  $\{H\}_{\circ Z}$  has scenario  $\psi$ .  $\square$

#### 4.6. Input/output relations

Given an  $(X, Y)$ -input/output diagram  $\mathcal{D}$ , define the *input/output relation*  $\text{Reln}(\mathcal{D})$  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  $\text{Reln}(\mathcal{D})$  is

- (total) for all  $\bar{x} \in \bar{X}$ , there exists  $\bar{y} \in \bar{Y}$  such that  $\bar{x}:\bar{y} \in \text{Reln}(\mathcal{D})$ :
- (monotone) If  $\bar{x}:\bar{y} \in \text{Reln}(\mathcal{D})$ , and  $\bar{x} \leq \bar{x}'$ , then there exists  $\bar{y}'$ , with  $\bar{y} \leq \bar{y}'$ , such that  $\bar{x}':\bar{y}' \in \text{Reln}(\mathcal{D})$ .

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

**Proof.** Straightforward from Lemma 4.3.1.  $\square$

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

$$\{y \in Y : \exists x \leq \bar{x}, x:y \in R_{\text{fin}}\},$$

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

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

**Proof.** Define the machine  $M$  as follows:

- Proper states: all elements  $x:y \in R_{\text{fin}}$ . Take  $\varepsilon_X:\varepsilon_Y$  as the start state.
- Proper transitions: all pairs  $(x:y, v) \in R_{\text{fin}} \times Y$ , such that  $x:yv \in R_{\text{fin}}$ . Define  $\text{dom}(x:y, v) = x:y$  and  $\text{cod}(x:y, v) = x:yv$ . Take the transitions  $(x:y, \varepsilon_Y)$  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  $M$  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  $\bar{x}$  of  $X$ , and  $\pi_Y(H)$  is a maximal directed subset of

$$\{y \in Y : \exists x \leq \bar{x}, x:y \in R_{\text{fin}}\}.$$

Since  $R$  is continuous, it follows that  $M$  has  $R$  as its input/output relation.  $\square$

**Lemma 4.6.3.** Suppose  $\phi$  is a total  $(Z \times X, Y \times Z)$ -Kahn function with  $\phi(\bar{z}:\bar{x}) = \bar{y}:\bar{z}$ , and  $\psi$  is the restriction of  $\phi$  to prefixes of  $\bar{z}:\bar{x}$ . Then  $\psi$  is feedback-compatible, and  $\{\psi\}_{\circ Z}$  is the restriction of  $\{\phi\}_{\circ Z}$  to prefixes of  $\bar{x}$ .

**Proof.** The fact that  $\psi$  is feedback-compatible follows immediately from the monotonicity of  $\phi$  and the assumption that  $\phi(\bar{z}:\bar{x}) = \bar{y}:\bar{z}$ . A simple induction then shows that for each  $k \geq 0$  the scenario  $\psi^{(k)}$  is the restriction to prefixes of  $\bar{x}$  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.  $\square$

**Lemma 4.6.4.** Suppose  $\phi$  is a total  $(Z \times X, Y \times Z)$ -Kahn function. Then  $\{\phi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$  iff the restriction  $\psi$  of  $\phi$  to prefixes of  $\bar{z}:\bar{x}$  is feedback-compatible, and  $\{\psi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ .

**Proof.** Suppose  $\{\phi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ . Let  $\psi$  be the restriction of  $\phi$  to prefixes of  $\bar{z}:\bar{x}$ . Then  $\phi(\bar{z}:\bar{x}) = \bar{y}:\bar{z}$ , so by Lemma 4.6.3  $\psi$  is feedback-compatible and  $\{\psi\}_{\circ Z}$  is the restriction of  $\{\phi\}_{\circ Z}$  to prefixes of  $\bar{x}$ . Hence  $\{\psi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ .

Conversely, suppose the restriction  $\psi$  of  $\phi$  to prefixes of  $\bar{z}:\bar{x}$  is feedback-compatible, and that  $\{\psi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ . Then  $\phi(\bar{x}:\bar{z}) = \bar{y}:\bar{z}$ , so by Lemma 4.6.3,  $\{\psi\}_{\circ Z}$  is the restriction to prefixes of  $\bar{x}$  of  $\{\phi\}_{\circ Z}$ . Since  $\{\psi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ , it follows that  $\{\phi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ .  $\square$

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

$$\text{Fun}(\{\mathcal{D}\}_{=Z}) = \{\text{Fun}(\mathcal{D})\}_{\circ Z}.$$

**Proof.** By definition of  $\text{Fun}$ ,  $\text{Fun}(\{\mathcal{D}\}_{=Z})(\bar{x}) = \bar{y}:\bar{z}$  iff there exists a scenario  $\psi$  of  $\{\mathcal{D}\}_{=Z}$  with complete trace  $\bar{x}:\bar{y}:\bar{z}$ . By Theorem 4.6, this is true iff there exists a scenario  $\phi$  of  $\mathcal{D}$ , with complete trace  $\bar{z}:\bar{x}:\bar{y}:\bar{z}$ , such that  $\{\phi\}_{\circ Z}$  has complete trace  $\bar{x}:\bar{y}:\bar{z}$ . By Lemma 4.6.1, this is true iff there exists a scenario  $\phi$ , such that  $\phi$  is the restriction of  $\text{Fun}(\mathcal{D})$  to prefixes of  $\bar{z}:\bar{x}$ ,  $\phi(\bar{z}:\bar{x}) = \bar{y}:\bar{z}$ , and  $\{\phi\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ . By Lemma 4.6.4, this is true iff  $\{\text{Fun}(\mathcal{D})\}_{\circ Z}(\bar{x}) = \bar{y}:\bar{z}$ .  $\square$

## 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 structural 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, an attempt was made to solve these problems using a model of processes based on ordinary (nondeterministic) labeled transition systems. There were two difficulties that seemed inherent in such a model. 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  $\lambda$ -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  $\lambda$ -calculus work [6, 27], and the extension of this work to term-rewriting systems [18], 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 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 [47] defines the notion of a “synchronization tree,” which is a (possibly infinite) tree whose arcs are labeled with elements of a “synchronization algebra.” In [46], labeled *event structures* [34] 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  $\epsilon$ , and Winskel’s operation  $\bullet$  with our operation  $v$ . 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  $(c, q_0)$ , it is straightforward to make the set of events  $(C, q_0)$  into an event structure by defining the “consistent” sets of events to be the finite sets that are consistent in the sense we have defined here, and defining a consistent set  $T$  to “enable” an event  $t$  iff there is a subset  $U$  of  $T$  such that  $t \uparrow (\vee U)$  is a nonidentity atom. 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 [30] use ideas from multilinear algebra to model nondeterministic and concurrent processes without iteration or recursion. An important role is played by “positive semirings,” whose formal properties are closely related to 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  $\delta: Q \times X \rightarrow Q$  to the notion of a “dynamics,” which is a morphism  $\delta: X(Q) \rightarrow Q$ , where  $Q$  is an object of an arbitrary category  $K$ , and  $X$  is an endofunctor of  $K$ . Arbib and Manes’ theory is applied to “port automata” in [42]. 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 CTS to achieve this goal. The product-forming functor  $(-\times X)$  does not yield a general enough class of dynamics.

The work of Winkowski [44, 45], is motivated by considerations in the theory of Petri nets. In [44], 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 [40] define a dataflow-like model in which a process is represented by 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 [26] 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 or colimit of a diagram generated by repeated application of a suitably continuous endofunctor on  $\text{Mach}_{X,Y}$ . 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 [35]. 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  $\text{Proc}_{X,Y}$ , 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. The recent work [5] is relevant here. 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” [36, 33], which is fundamental for ordinary transition systems, might be reasonably generalized to CTS’s.

Finally, the full abstraction problem for machines remains intriguing. 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, and by proving rigorously that this model must lie somewhere between scenario sets and input/output relations in information content. The detailed information about the feedback operation we have obtained here seems likely to be useful in ultimately resolving this important question.

### Acknowledgment

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*, Lecture Notes in Computer Science **158** (Springer-Verlag, New York, 1983) 1-13.
- [2] M.A. Arbib and E.G. Manes, *Arrows, Structures, and Functors: The Categorical Imperative* (Academic Press, New York, 1975).
- [3] R.J. Back and N. Mannila, A refinement of Kahn's semantics to handle nondeterminism and communication, in: *Proc. ACM Symp. on Principles of Distributed Computing* (1982) 111-120.
- [4] H.F. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics **103** (North-Holland, Amsterdam, 1981).
- [5] M. Bednarczyk, Categories of asynchronous systems, Ph.D. Thesis, University of Sussex, October 1987.
- [6] G. Berry and J.-J. Lévy, Minimal and optimal computations of recursive programs, *J. ACM* **26**(1) (1979) 148-175.
- [7] J.D. Brock, A formal model of non-determinate dataflow computation, Ph.D. Thesis, Massachusetts Institute of Technology, 1983, available as MIT/LCS/TR-309.
- [8] J.D. Brock and W.B. Ackerman, Scenarios: a model of non-determinate computation, in: *Formalization of Programming Concepts*, Lecture Notes in Computer Science **107** (Springer-Verlag, New York, 1981) 252-259.
- [9] S.D. Brookes, On the relationship of ccs and csp, in: *Proc. ICALP 83* (Springer-Verlag, New York, 1983) 83-96.
- [10] S.D. Brookes and W.C. Rounds, Behavioral equivalence relations induced by programming logics, in: *Proc. of ICALP 83* (Springer-Verlag, New York, 1983) 97-108.
- [11] R. de Nicola and M.C.B. Hennessy, Testing equivalences for processes, *Theoret. Comput. Sci.* **34**(1) (1984) 83-133.
- [12] S. Eilenberg, *Automata, Languages, and Machines Vol. A* (Academic Press, New York, 1974).
- [13] A.A. Faustini, An operational semantics for pure dataflow, in: *Proc. Internat. Coll. on Automata, Languages, and Programming*. Lecture Notes in Computer Science **140** (Springer-Verlag, New York, 1982) 212-224.
- [14] I. Guessarian, *Algebraic Semantics*, Lecture Notes in Computer Science **99** (Springer-Verlag, New York, 1981).
- [15] J. Hartmanis and R.E. Stearns, *Algebraic Structure Theory of Sequential Machines* (Prentice-Hall, New York, 1966).
- [16] H. Herrlich and G.E. Strecker, *Category Theory*, Sigma Series in Pure Mathematics (Heldermann Verlag, Berlin, 1979).
- [17] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21**(8) (1978) 666-676.
- [18] G. Huet, Formal structures for computation and deduction, Unpublished manuscript, INRIA, France, 1986.
- [19] G. Kahn, The semantics of a simple language for parallel programming, in: J.L. Rosenfeld, ed., *Information Processing 74* (North-Holland, Amsterdam, 1974) 471-475.

- [20] G. Kahn and D.B. MacQueen, Coroutines and networks of parallel processes, in: B. Gilchrist, ed., *Information Processing 77* (North-Holland, Amsterdam, 1977) 993–998.
- [21] R.M. Keller, Denotational models for parallel programs with indeterminate operators, in: E.J. Neuhold, ed., *Formal Description of Programming Concepts* (North-Holland, Amsterdam, 1978) 337–366.
- [22] R.M. Keller, Formal verification of parallel programs, *Comm. ACM* 19(7) (1976) 371–384.
- [23] R.M. Keller and P. Panangaden, Semantics of networks containing indeterminate operators, in: *Seminar on Concurrency*, Lecture Notes in Computer Science 197 (Springer-Verlag, New York, 1984) 479–496.
- [24] J.N. Kok, Denotational semantics of nets with nondeterminism, in: *ESOP 86*, Lecture Notes in Computer Science 213 (Springer-Verlag, New York, 1986) 237–249.
- [25] J.N. Kok, A fully abstract semantics for data flow nets, in: *Lecture Notes in Computer Science* 259 (Springer-Verlag, New York, 1987) 351–368.
- [26] A. Labella and A. Pettorossi, Categorical models for handshaking communications, in: *Annales Societatis Mathematicae Polonae. SERIES IV: Fundamenta Informaticae* (1985).
- [27] J.-J. Lévy, Réductions correctes et optimales dans le lambda calcul, Ph.D. Thesis, Université de Paris VII, 1978.
- [28] S. Mac Lane, *Categories for the Working Mathematician*, Graduate Texts in Mathematics 5 (Springer-Verlag, New York, 1971).
- [29] D.B. MacQueen, Models for distributed computing, Technical Report 351, INRIA, 1979.
- [30] M.G. Main and D.B. Benson, Functional behavior of nondeterministic and concurrent programs, *Inform. and Contr.* 62 (1984) 144–189.
- [31] A. Mazurkiewicz, Trace theory, in: *Advanced Course on Petri Nets* (GMD, Bad Honnef, 1986).
- [32] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92 (Springer-Verlag, New York, 1980).
- [33] R. Milner, Lectures on a calculus for communicating systems, in: *Seminar on Concurrency*, Lecture Notes in Computer Science 197 (Springer-Verlag, New York, 1984) 197–220.
- [34] M. Nielsen, G. Plotkin and G. Winskel, Petri nets, event structures, and domains, Part I, *Theoret. Comput. Sci.* 13 (1981) 85–108.
- [35] P. Panangaden and E.W. Stark, Computations, residuals, and the power of indeterminacy, in: *Proc. ICALP 1988*, Lecture Notes in Computer Science 317 (Springer-Verlag, New York, 1988) 439–454.
- [36] D.M.R. Park, Concurrency and automata on infinite sequences, in: *Theoretical Computer Science*, Lecture Notes in Computer Science 104 (Springer-Verlag, New York, 1981).
- [37] D.M.R. Park, The “fairness problem” and nondeterministic computing networks, in: *Proc. 4th Advanced Course on Theoretical Computer Science* (Mathematisch Centrum, Amsterdam, 1982) 133–161.
- [38] V.R. Pratt, On the composition of processes, in: *Proc. 9th Annual ACM Symp. on Principles of Programming Languages* (1982) 213–223.
- [39] V.R. Pratt, The pomset model of parallel processes: unifying the temporal and the spatial, in: *Seminar on Concurrency*, Lecture Notes in Computer Science 197 (Springer-Verlag, New York, 1984) 180–196.
- [40] J. Staples and V.L. Nguyen, A fixpoint semantics for nondeterministic data flow, *J. ACM* 32(2) (1985) 411–444.
- [41] E.W. Stark, Concurrent transition system semantics of process networks, in: *Proc. 14th ACM Symp. Principles of Programming Languages* (January 1987) 199–210.
- [42] M. Steenstrup, M.A. Arbib and E.G. Manes, Port automata and the algebra of concurrent processes, *J. Comput. System Sci.* 27(1) (1983) 29–50.
- [43] P.S. Thiagarajan, Elementary net systems, in: *Advanced Course on Petri Nets* (GMD, Bad Honnef, September 1986).
- [44] J. Winkowski, An algebraic description of system behaviors, *Theoret. Comput. Sci.* 21 (1982) 315–340.
- [45] J. Winkowski, Behaviors of concurrent systems, *Theoret. Comput. Sci.* 12 (1980) 39–60.
- [46] G. Winskel, Event structures, in: *Advanced Course on Petri Nets* (GMD, Bad Honnef, 1986).
- [47] G. Winskel, Synchronization trees, *Theoret. Comput. Sci.* 34 (1984) 33–82.