Connections between a Concrete and an Abstract Model of Concurrent Systems

Eugene W. Stark*

Department of Computer Science

State University of New York at Stony Brook

Stony Brook, NY 11794 USA

Abstract

We define a concrete operational model of concurrent systems, called *trace automata*. For such automata, there is a natural notion of *permutation equivalence* of computation sequences, which holds between two computation sequences precisely when they represent two interleaved views of the "same concurrent computation." Alternatively, permutation equivalence can be characterized in terms of a *residual operation* on transitions of the automaton, and many interesting properties of concurrent computations can be expressed with the help of this operation. In particular, concurrent computations, ordered by "prefix," form a Scott domain whose structure we characterize up to isomorphism.

By axiomatizing the properties of the residual operation, we obtain a more abstract formulation of automata, which we call *concurrent transition systems* (CTS's). By exploiting a correspondence between concurrent alphabets and certain CTS's, we are able to use the rich algebraic structure of CTS's to obtain results in trace theory. Finally, we connect CTS's and trace automata by obtaining a characterization of those CTS's that correspond in a natural way to trace automata, and we show how the correspondence suggests an interesting notion of morphism of trace automata.

1 Introduction

Labeled transition systems (LTS's) have been used frequently as an operational semantics of concurrent processes. In typical formulations, an LTS is a tuple $A = (E, Q, T, *)$, where E is a set of *events*, Q is a set of *states*, $* \in Q$ is a distinguished *start state*, and $T \subseteq Q \times E \times Q$ is a set of *transitions*, which represent potential computation steps. Although useful for many applications, LTS's are not ideally suited as a model of concurrency, since they contain no mathematical structure with which

*Research supported in part by NSF Grant CCR-8702247.

concurrency can be represented and reasoned about directly. Instead, concurrency in computations must be represented somewhat artificially by interleaving, and reasoning about concurrency requires that we make use of auxiliary information (*e.g.* which pairs of transitions "commute") not explicitly formalized in the LTS model.

In an effort to get explicit concurrency information into the definition of a transition system, we might introduce a symmetric, irreflexive *concurrency relation* $\|$ on E, and require that the transitions respect this concurrency information in a suitable sense. Although there is some flexibility in the exact sense in which concurrency is to be respected by the transitions, the end result is essentially the class of *trace automata* which we define below. This kind of automaton, which arises naturally in the study of trace theory [1, 11], has been the subject of investigation by Bednarczyk [2], Kwiatkowska [8], and Shields [13], and has been used by the author to study nondeterministic dataflow networks [12, 17]. The familiar mapping that takes a finite computation sequence of an automaton to the string it generates is now replaced by a monotone mapping from the prefix-ordered set of finite computation sequences to the prefix-ordered set of *traces*, which are equivalence classes of strings modulo a congruence relation induced by the concurrency relation.

We can actually go a bit further than this. If we regard computation sequences having the same trace as "equivalent interleaved views" of a single concurrent computation, and we factor the poset of computation sequences by this equivalence, then the trace mapping becomes an isomorphism between the resulting poset of "finite concurrent computations" and the "trace language" generated by the automaton, where the latter is viewed as a subset of the prefix-ordered set of all traces. Ideal completion of the poset of finite concurrent computations results in a Scott domain containing both finite and infinite concurrent computations. The domain of concurrent computations is much more interesting than the poset of finite computation sequences, since concurrency is reflected in the former through the existence of nontrivial upper bounds. Since our goal is to make concurrency explicit, one might argue that concurrent computations, rather than computation sequences, ought to be the main focus of attention.

One of our main results is the following characterization of the structure of the domains of concurrent computations of trace automata:

> The domain of concurrent computations of a trace automaton is isomorphic to a normal subdomain U of the domain $\bar{E}$ of traces generated by the event set E and concurrency relation $\|$, where the inclusion of U in $\bar{E}$ preserves prime intervals. Conversely, if U is a normal subdomain of a domain $\bar{E}$ of traces, such that the inclusion of U in $\bar{E}$ preserves prime intervals, then U is isomorphic to the domain of concurrent computations of a trace automaton.

The proof of this characterization theorem uses in an essential way the observation that equivalence of computations can be described, independently of trace theory, using the concept of a "residual operation" on computation sequences. Intuitively, taking the residual of a computation sequence γ "after" a computation sequence δ corresponds to "cancelling from γ the greatest common prefix, up to concurrency, of

γ and δ." The computation sequences γ and δ are equivalent precisely when each is completely cancelled by the other. Residuals have previously been used by Lévy [9] in the study of the λ-calculus, to define the notion of strongly equivalent reductions. The same ideas can also be applied [3, 4, 7] to the study of recursive programs and left-linear term-rewriting systems without critical pairs. In that work, residuals are used to keep track of what happens to one redex in a term while other redexes are contracted. Our use here is analogous: the residual operation allows us to keep track of what happens to one enabled transition in a system while other concurrent transitions are executed. Boudol and Castellani [5] have also exploited the use of residuals and permutation equivalence in reasoning about concurrency.

By axiomatizing the properties of a residual operation necessary to obtain the equivalence relation on computation sequences, we arrive at the definition of *concurrent transition systems* (CTS's) [15, 16]. The defining axioms generate a rich algebraic theory, which we have found to be of use in the study of concurrent systems [12, 15, 17]. A suitable definition of morphism makes the class of all CTS's into a category **CTS**, which has small limits, small coproducts, small filtered colimits, and is cartesian closed. Moreover, many interesting constructions on automata have universal or couniversal characterizations either in **CTS** or in functor categories built from it. Included among these constructions are those that extract the computational behavior of a CTS.

In this paper, we give the details of the story outlined above. Our goal is to motivate explicitly the connection, between the concrete, easily understood trace automaton model, and the more abstract concurrent transition systems which the author has described elsewhere [15, 16]. To complete this connection between abstract and concrete, we exhibit properties that characterize up to isomorphism those CTS's that are derived from "event automata."

2 Trace Automata

In this paper, all sets whose cardinality is left unspecified are assumed to be at most countable.

A *concurrent alphabet* is a set E, equipped with a symmetric, irreflexive binary relation $\|_E$, called the *concurrency relation.*

A *trace automaton* (henceforth simply "automaton") is a tuple $A = (E, Q, T)$, where

- E is a concurrent alphabet, whose elements are called *events.* We assume that E does not contain the special symbol ϵ, called the *identity event.*
- Q is a set of *states.*
- $T \subseteq Q \times (E \cup \{\epsilon\}) \times Q$ is a set of *transitions.* We usually write $t : q \xrightarrow{a} r$, or just $q \xrightarrow{a} r$, to denote a transition $t = (q, a, r)$ in T.

These data are required to satisfy the following conditions:

(Identity) $q \xrightarrow{\epsilon} r$ iff $q = r$.

(Disambiguation) If $q \xrightarrow{a} r$ and $q \xrightarrow{a} r'$, then $r = r'$.

(Commutativity) For all states q and events a, b, if $a \|_E b$, $q \xrightarrow{a} r$, and $q \xrightarrow{b} s$, then for some state p there exist transitions $s \xrightarrow{a} p$ and $r \xrightarrow{b} p$.

A trace automaton *with start state* is a tuple $(E, Q, T, *)$, where (E, Q, T) is a trace automaton, and $* \in Q$ is a distinguished state.

Intuitively, if $a \in E$, then a transition $q \xrightarrow{a} r$ represents a potential computation step of A in which event a occurs and the state changes from q to r. *Identity transitions* $\mathrm{id}_q = (q \xrightarrow{\epsilon} q)$ do not represent steps of A. Rather, these transitions play a purely technical role, which will become evident when we define the notion of a "residual operation" below. We say that event $a \in E$ is *enabled* in state q if there exists a transition $q \xrightarrow{a} r$. By the disambiguation condition, if $q \xrightarrow{a} r$, then r is uniquely determined by q and a. If $t : q \xrightarrow{a} r$, then q is called the *domain* $\mathrm{dom}(t)$ of t and r is called the *codomain* $\mathrm{cod}(t)$ of t. Transitions t and u are called *coinitial* if $\mathrm{dom}(t) = \mathrm{dom}(u)$.

A *finite computation sequence* for an automaton is a finite sequence γ of non-identity transitions of the form:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n.$$

The number n is called the *length* $|\gamma|$ of γ. By convention, we regard an identity transition id_q. as identical to the computation sequence of length zero from state q. An *infinite computation sequence* is an infinite sequence of non-identity transitions:

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots.$$

We extend notation and terminology for transitions to computation sequences, so that if γ is a computation sequence, then the *domain* $\mathrm{dom}(\gamma)$ of γ is the state q_0, and if γ is finite, then the *codomain* $\mathrm{cod}(\gamma)$ of γ is the state q_n. We write $\gamma : q \to r$ to assert that γ is a finite computation sequence with domain q and codomain r. A computation sequence γ is *initial* if $\mathrm{dom}(\gamma)$ is the distinguished start state $*$. If $\gamma : q \to r$ and $\delta : q' \to r'$ are finite computation sequences, then γ and δ are called *composable* if $q' = r$, and we define their *composition* to be the finite computation sequence $\gamma\delta : q \to r'$, obtained by concatenating γ and δ and identifying $\mathrm{cod}(\gamma)$ with $\mathrm{dom}(\delta)$. The operation of composition of finite computation sequences is associative, and identity transitions (computation sequences of length 0) behave as units for it. A finite computation sequence γ is a *prefix* of a computation sequence δ, and we write $\gamma \leq \delta$, iff there exists a computation sequence ξ with $\gamma\xi = \delta$.

Trace automata (with start state) are nearly identical to the "forward stable asynchronous systems" of Bednarczyk [2]; the difference being that we retain his forward stability axiom (our commutativity axiom), but we omit his axiom stating that if $q \xrightarrow{a} r$, $r \xrightarrow{b} s$, and $a \|_E b$, then for some state p there exist transitions $q \xrightarrow{b} p$ and $p \xrightarrow{a} s$. Although Bednarczyk seems to treat this axiom as more fundamental than forward stability, it seems overly restrictive, and we shall see that much can be done without it.

Trace automata can be used as the basis for an operational model of nondeterministic dataflow networks, which consist of a collection of concurrently and asynchronously executing processes that communicate by passing "value tokens" over named "ports." This is done by introducing additional structure to distinguish between "input events," "output events," and "internal events," and then requiring that input and output events have a specific form that reflects the port structure. We give below some of the definitions to illustrate how this is done. The reader wishing further discussion is referred to [10, 12, 15, 17].

An *input/output* automaton is a triple (A, X, Y), where $A = (E, Q, T, *)$ is a trace automaton with start state, and X, Y are disjoint subsets of E, called the sets of *input events* and *output events*, respectively. Elements of $E \setminus (X \cup Y)$ are called *internal events*. The following property is required to hold:

(Receptivity) For all states q and input events a, event a is enabled in state q.

A *port automaton* is an input/output automaton (A, X, Y) equipped with a set V of *values*, a set I of *input ports*, and a set O of *output ports*, such that $X = I \times V$, $Y = O \times V$, and such that whenever (p, v) and (p', v') are events in $X \cup Y$, then $(p, v) \|_E (p', v')$ iff $p = p'$.

Two particularly well-behaved classes of input/output automata are the "monotone" automata and the "determinate" automata. An input/output automaton is *monotone* if the following additional property holds:

(Monotonicity) $a \|_E b$ whenever $a \in X$ and $b \in E \setminus X$.

Intuitively, the monotonicity property states that the arrival of input cannot disable any enabled output or internal transitions, since if b is an output event enabled in state q, and if a is an arbitrary input event, then $a \|_E b$ by monotonicity, $q \xrightarrow{a} r$ for some r by receptivity, hence b is enabled in state r by commutativity. An automaton is *determinate* if it satisfies the following condition:

(Determinacy) Suppose $q \xrightarrow{b} r$ and $q \xrightarrow{c} s$, where b and c are distinct non-input events. Then $b \|_E c$.

It can be shown [15, 16] that determinate input/output automata have functional input/output behavior.

2.1 Concurrent Computations

A partially ordered set $(D, \sqsubseteq)$ is *consistently complete* if each pair of elements of D that have an upper bound, have a least upper bound. A (Scott) *domain* is an ω-algebraic, consistently complete CPO $D = (D, \sqsubseteq, \perp)$. A domain D is *finitary* if for all finite (=isolated=compact) elements $d \in D$ the set $\{d' \in D : d' \sqsubseteq d\}$ is finite. An *atom* of D is a minimal non-$\perp$ element of D. If D and E are domains, then a monotone map $f : D \to E$ is *continuous* if it preserves suprema of ω-chains, *strict* if $f(\bot_D) = \bot_E$, and *additive* if whenever d, d' are consistent elements of D, then $f(d), f(d')$ are consistent elements of E, and $f(d \sqcup d') = f(d) \sqcup f(d')$. The map f *reflects consistency* if whenever $f(d)$ and $f(d')$ are consistent elements of E, then d and d' are consistent elements of D.

The set of all finite and infinite initial computation sequences of a trace automaton with start state $A = (E, Q, T, *)$, forms a domain when equipped with the prefix ordering $\leq$. This domain is of limited utility, since it does not take into account any concurrency information. However, there is a natural notion of "permutation equivalence" of computations of A, which captures the idea of "equivalent interleaved views" of the same concurrent computation. By factoring the domain of initial computation sequences by permutation equivalence we obtain a more useful and interesting domain of "concurrent computations."

Formally, define permutation equivalence to be the least congruence $\sim$, respecting concatenation, on the set of finite computation sequences of A such that:

- Computation sequences $q \xrightarrow{a} r \xrightarrow{b} p$ and $q \xrightarrow{b} s \xrightarrow{a} p$ are $\sim$-related if $a \|_E b$.

Closely related to permutation equivalence is the *permutation preorder* relation $\precsim$ on finite computation sequences of A, which is defined to be the transitive closure of $(\leq \cup \sim)$. It is not difficult to see that $\gamma \sim \delta$ iff $\gamma \precsim \delta$ and $\delta \precsim \gamma$.

Permutation preorder extends in a straightforward way to infinite computation sequences as well: if γ' and δ' are coinitial finite or infinite computation sequences, then define $\gamma' \precsim \delta'$ to hold iff for every finite $\gamma \leq \gamma'$ there exists a finite $\delta \leq \delta'$, such that $\gamma \precsim \delta$. We may then extend permutation equivalence to infinite computation sequences by defining $\gamma' \sim \delta'$ iff $\gamma' \precsim \delta'$ and $\delta' \precsim \gamma'$.

2.1.1 Permutation Preorder and Traces

Because the concurrency in a trace automaton is completely determined by the concurrency relation on events, we can describe permutation preorder as the preorder induced by a certain mapping from computation sequences to a domain of "traces." To state this formally, we need some basic definitions from trace theory [1, 2, 11].

Suppose E is a concurrent alphabet. Let E^* denote the free monoid generated by E, then there is a least congruence $\sim_E$ on E^* such that $a \|_E b$ implies $ab \sim_E ba$ for all $a, b \in E$. The quotient $E^* / \sim_E$ is the *free partially commutative monoid* generated by E, and its elements are called *traces*. We use ϵ to denote the monoid identity, and if $x \in E^*$, then we use $[x]$ to denote the corresponding element of $E^* / \sim_E$. Define the relation $\sqsubseteq$ on the monoid $E^* / \sim_E$ by: $[x] \sqsubseteq [y]$ iff $\exists [z]([x][z] = [y])$. It is not difficult to show that $\sqsubseteq$ is a partial order, with ϵ as a least element. Let $\bar{E}$ denote the ideal completion of this partial order, then $\bar{E}$ is an algebraic CPO whose finite elements are the principal ideals generated by elements of $E^* / \sim_E$. We call $\bar{E}$ the *domain of traces generated by* the concurrent alphabet E. (This terminology is justified by Lemma 2.3, which shows that $\bar{E}$ is consistently complete, hence a domain. For the moment, we only need the fact that $\bar{E}$ is an algebraic CPO.) Notice that since the finite elements of $\bar{E}$ are in bijective correspondence with the elements of $E^* / \sim_E$, they inherit the monoid operation of $E^* / \sim_E$, with the least element of $\bar{E}$ as the monoid identity. In the sequel, we identify elements of $E^*/{\sim}$ with the corresponding finite elements of $\bar{E}$.

Now, suppose

$$\gamma = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$$

is a finite or infinite computation sequence of an automaton $A = (E, Q, T)$. Define the *trace* of γ to be the element $\mathrm{tr}(\gamma) = \bigsqcup_{k \geq 0} [a_1 a_2 \ldots a_k]$ of the domain of traces $\bar{E}$. Obvious consequences of this definition are: (1) $\mathrm{tr}(\mathrm{id}_q) = \epsilon$, (2) if γ and δ are composable finite computation sequences, then $\mathrm{tr}(\gamma\delta) = \mathrm{tr}(\gamma)\mathrm{tr}(\delta)$, and (3) the map tr is continuous, with respect to the prefix ordering $\leq$ on computation sequences and the ordering $\sqsubseteq$ on traces.

Theorem 1 *Suppose γ and δ are coinitial computation sequences. Then $\gamma \precsim \delta$ holds iff* $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$.

Proof – We first prove the result for the special case that γ and δ are finite. Suppose $\gamma \precsim \delta$. Then there exists a finite sequence

$$\gamma = \xi_0, \xi_1, \ldots, \xi_n = \delta,$$

such that for each k with $0 \leq k < n$, one of the following two relationships holds:

1. $\xi_k \leq \xi_{k+1}$.
2. ξ_{k+1} is obtained from ξ_k by replacing a subsequence of the form $q \xrightarrow{a} r \xrightarrow{b} p$, where $a \|_E b$, by a sequence $q \xrightarrow{b} s \xrightarrow{a} p$.

In case (1), $\mathrm{tr}(\xi_k) \sqsubseteq \mathrm{tr}(\xi_{k+1})$, and in case (2), $\mathrm{tr}(\xi_k) = \mathrm{tr}(\xi_{k+1})$. Hence $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$ by reflexivity and transitivity of $\sqsubseteq$.

Conversely, suppose $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$. Let x and y be the sequences of events appearing in γ and δ, respectively. Then $[x] = \mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta) = [y]$, so by definition of $\sqsubseteq$ there exists z such that $xz \sim y$. It follows that the string y can be transformed into the string xz by a finite sequence of steps in which adjacent pairs of concurrent symbols are permuted. By performing the same sequence of permutation steps starting with δ, we obtain a proof that $\delta \sim_E \gamma\xi$ for some computation sequence ξ.

We now extend the result to include infinite computation sequences. Suppose $\gamma' \precsim \delta'$, where γ' and δ' are arbitrary. Given any finite $[x] \sqsubseteq \mathrm{tr}(\gamma')$, by the continuity of the map tr with respect to the prefix ordering $\leq$, there exists a finite $\gamma \leq \gamma'$ such that $[x] \sqsubseteq \mathrm{tr}(\gamma)$. Choose a finite $\delta \leq \delta'$ such that $\gamma \precsim \delta$, then $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$ by the finite case of the theorem. Thus, for each finite $[x] \sqsubseteq \mathrm{tr}(\gamma')$ there exists a finite $[y] \sqsubseteq \mathrm{tr}(\delta')$ such that $[x] \sqsubseteq [y]$. By the fact that the CPO $\bar{E}$ is algebraic, it follows that $\mathrm{tr}(\gamma') \sqsubseteq \mathrm{tr}(\delta')$.

Conversely, suppose $\mathrm{tr}(\gamma') \sqsubseteq \mathrm{tr}(\delta')$. Given a finite $\gamma \leq \gamma'$, let $[x] = \mathrm{tr}(\gamma)$. Then $[x] \sqsubseteq \mathrm{tr}(\gamma')$ so by algebraicity of $\bar{E}$ we may choose a finite $[y] \sqsubseteq \mathrm{tr}(\delta')$ such that $[x] \sqsubseteq [y]$. By continuity of tr with respect to the prefix ordering $\leq$ on computation sequences, there exists a finite $\delta \leq \delta'$ such that $[y] \sqsubseteq \mathrm{tr}(\delta)$. But then $\gamma \precsim \delta$ by the finite case of the theorem. ∎

2.1.2 Permutation Preorder and Residuals

The permutation preorder can also be characterized in a much different, and ultimately more useful way, using the notion of the "residual" of one finite computation sequence "after" another. Residuals, previously used for the λ-calculus and term-rewriting systems [3, 4, 7, 9], have been shown in [12, 15, 16] to be extremely useful in reasoning about concurrent systems. Here, we formalize the notion of residual as a partial binary operation $\uparrow$ on finite computation sequences, such that $\gamma \uparrow \delta$ (read γ "after" δ) is defined, and γ and δ are said to be *consistent*, exactly when γ and δ are coinitial finite computation sequences that could both be part of the "same concurrent computation." In general, δ will then contain some transitions that "overlap" with γ and some transitions that are "concurrent" with γ, and the residual $\gamma \uparrow \delta$ is defined to be what is left of γ after the part of δ that overlaps with it has been "cancelled." The residual $\gamma \uparrow \delta$ is undefined when γ contains some nondeterministic choice that is incompatible with a choice made in δ. In this case we say that γ and δ *conflict*. Observe that we distinguish between two types of choice that may be represented in an automaton: *concurrent* choice, in which events a, b with $a \| b$ are both enabled in the same state q, and *nondeterministic* choice, in which a and b are both enabled in state q but we do not have $a \| b$.

We first define the residual operation for coinitial computation sequences $\gamma : q \to r$ and $\delta : q \to s$ of length ≤ 1. There are three cases:

1. If $\gamma = \mathrm{id}_q$, then $\gamma \uparrow \delta = \mathrm{id}_s$ and $\delta \uparrow \gamma = \delta$.
2. If γ is a non-identity transition $q \xrightarrow{a} r$, and δ is a non-identity transition $q \xrightarrow{a} s$, then $r = s$ by the disambiguation condition. Define $\gamma \uparrow \delta = \mathrm{id}_s = \mathrm{id}_r = \delta \uparrow \gamma$.
3. If γ is a non-identity transition $q \xrightarrow{a} r$, and δ is a non-identity transition $q \xrightarrow{b} s$, where $a \neq b$, then $\gamma \uparrow \delta$ and $\delta \uparrow \gamma$ are defined iff $a \|_E b$. In this case, the commutativity property implies there must exist transitions $s \xrightarrow{a} p$ and $r \xrightarrow{b} p$, which we take as $\gamma \uparrow \delta$ and $\delta \uparrow \gamma$, respectively.

Lemma 2.1 *The operation $\uparrow$ has the following properties, where γ, δ, and ξ denote computation sequences of length ≤ 1.*

1. *If $\gamma \uparrow \delta$ is defined, then so is $\delta \uparrow \gamma$, and we have $\mathrm{dom}(\gamma \uparrow \delta) = \mathrm{cod}(\delta)$, $\mathrm{dom}(\delta \uparrow \gamma) = \mathrm{cod}(\gamma)$, and $\mathrm{cod}(\gamma \uparrow \delta) = \mathrm{cod}(\delta \uparrow \gamma)$.*
2. *For all $\gamma : q \to r$, (a) $\mathrm{id}_q \uparrow \gamma = \mathrm{id}_r$; (b) $\gamma \uparrow \mathrm{id}_q = \gamma$; and (c) $\gamma \uparrow \gamma = \mathrm{id}_r$.*
3. *For all coinitial γ, δ, ξ, $(\xi \uparrow \gamma) \uparrow (\delta \uparrow \gamma) = (\xi \uparrow \delta) \uparrow (\gamma \uparrow \delta)$, whenever either side is defined.*
4. *For all coinitial $\gamma : q \to r$ and $\delta : q \to s$, if $\gamma \uparrow \delta = \mathrm{id}_s$ and $\delta \uparrow \gamma = \mathrm{id}_r$, then $\gamma = \delta$.*

Moreover, $\uparrow$ extends uniquely to an operation on finite computation sequences, in such a way that properties (1)-(3) are preserved and such that the following additional identities hold whenever either side is defined:

5. $\gamma \uparrow \delta\xi = (\gamma \uparrow \delta) \uparrow \xi$
$\delta\xi \uparrow \gamma = (\delta \uparrow \gamma)(\xi \uparrow (\gamma \uparrow \delta))$.

Proof – (1), (2), and (4) are obvious from the definitions. (3) is proved by a straightforward case analysis on γ, δ, and ξ (see [12]).

The extension of $\uparrow$ to finite computation sequences is done using (5) as a recursive definition. Verification that the resulting extension has properties (1), (2), and (3) is then done by induction on the lengths of the computation sequences involved. Property (4) does not necessarily hold for the extension; for example, because if $\gamma = q \xrightarrow{a} r \xrightarrow{b} p$ and $\delta = q \xrightarrow{b} s \xrightarrow{a} p$ with $a \|_E b$, then $\gamma \uparrow \delta = \mathrm{id}_p = \delta \uparrow \gamma$, but $\gamma \neq \delta$. ∎

We now derive the connection between residuals and permutation preorder. If γ and δ are coinitial, then define $\gamma \precsim \delta$ iff $\gamma \uparrow \delta$ is an identity.

Lemma 2.2 *The relation $\precsim$ is a preorder.*

Proof – Reflexivity holds because $\gamma \uparrow \gamma = \mathrm{id}_q$, where $q = \mathrm{cod}(\gamma)$. To show transitivity, suppose $\gamma \precsim \delta$ and $\delta \precsim \xi$. Then $\gamma \uparrow \delta$ is an identity, so $(\gamma \uparrow \delta) \uparrow (\xi \uparrow \delta)$ is an identity. Since $(\gamma \uparrow \delta) \uparrow (\xi \uparrow \delta) = (\gamma \uparrow \xi) \uparrow (\delta \uparrow \xi)$, it follows that $(\gamma \uparrow \xi) \uparrow (\delta \uparrow \xi)$ is an identity. But $\delta \uparrow \xi$ is an identity because $\delta \precsim \xi$, hence $\gamma \uparrow \xi$ is an identity. ∎

Theorem 2 *Suppose γ and δ are coinitial finite computation sequences. Then $\gamma \sqsubseteq \delta$ iff $\gamma \precsim \delta$.*

Proof – Suppose $\gamma \sqsubseteq \delta$. Then there exists a finite sequence

$$\gamma = \xi_0, \xi_1, \ldots, \xi_n = \delta,$$

such that for each k with $0 \leq k < n$, one of the following two relationships holds:

1. $\xi_k \leq \xi_{k+1}$.
2. ξ_{k+1} is obtained from ξ_k by replacing a subsequence of the form $q \xrightarrow{a} r \xrightarrow{b} p$, where $a \|_E b$, by a sequence $q \xrightarrow{b} s \xrightarrow{a} p$.

In case (1), $\xi_{k+1} = \xi_k \zeta$ for some ζ, so by Lemma 2.1, $\xi_k \uparrow \xi_{k+1} = (\xi_k \uparrow \xi_k) \uparrow \zeta$, which is an identity, hence $\xi_k \precsim \xi_{k+1}$. In case (2), $\xi_k = \zeta t u' \zeta'$ and $\xi_{k+1} = \zeta u t' \zeta'$, where t is the transition $q \xrightarrow{a} r$, u is the transition $q \xrightarrow{b} s$, $t' = t \uparrow u$, and $u' = u \uparrow t$. From this, a straightforward calculation using Lemma 2.1 shows that $\xi_k \uparrow \xi_{k+1}$ is an identity, so $\xi_k \precsim \xi_{k+1}$. Since $\xi_k \precsim \xi_{k+1}$ both in case (1) and in case (2), the result $\gamma \precsim \delta$ follows by the reflexivity and transitivity of $\precsim$.

Conversely, suppose $\gamma \precsim \delta$. Let $\xi = \delta \uparrow \gamma$, then γ, δ, and ξ are related as depicted in Figure 1, where the sides of each small diamond are single transitions, and the apex of each small diamond is obtained by applying the residual operation to the transitions forming its base. From this diagram, we may read off a proof that $\gamma\xi \sim \delta$. This is done by starting with the computation sequence $\gamma\xi$ represented by the path around the left-hand side of the large diamond, and, beginning with the

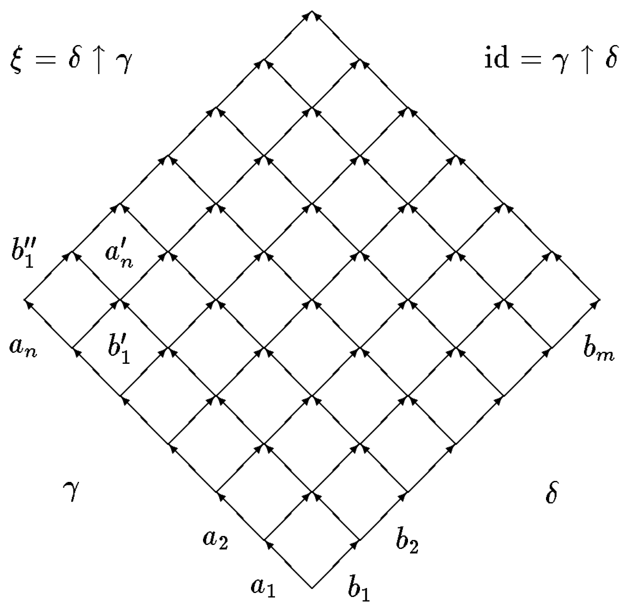

Figure 1: Residuals and Permutation Preorder

leftmost small diamond in the diagram, successively replacing the pairs of transitions forming the left-hand sides of the small diamonds by the pairs of transitions forming the right-hand sides. For example, in the first step we would replace $a_n b_1''$ by $b_1' a_n'$. Eventually, these replacement steps transform the sequence $\gamma\xi$ into the computation sequence represented by the path around the right-hand side of the large diamond; that is, into δ. ∎

Having characterized permutation preorder in terms of residuals, we may now obtain a great deal of information about the structure of the set of computation sequences, under the permutation preorder. The main result is Theorem 4 below. I do not know how to prove such a strong result without using residuals.

Lemma 2.3 *Suppose γ and δ are coinitial finite computation sequences. Then γ and δ have an upper bound with respect to $\precsim$ iff $\gamma \uparrow \delta$ is defined. Moreover, if $\gamma \uparrow \delta$ is defined, then γ and δ have a supremum with respect to $\precsim$, given by $\gamma(\delta \uparrow \gamma)$ or $\delta(\gamma \uparrow \delta)$, which are permutation equivalent.*

Proof – If γ and δ have an upper bound ξ with respect to $\precsim$, then $\gamma \uparrow \xi$ and $(\gamma \uparrow \xi) \uparrow (\delta \uparrow \xi)$ are identities. But $(\gamma \uparrow \xi) \uparrow (\delta \uparrow \xi) = (\gamma \uparrow \delta) \uparrow (\xi \uparrow \delta)$, hence $\gamma \uparrow \delta$ is defined.

Conversely, if $\gamma \uparrow \delta$ is defined, then since $\gamma \uparrow \gamma(\delta \uparrow \gamma) = (\gamma \uparrow \gamma) \uparrow (\delta \uparrow \gamma)$ and $\delta \uparrow \gamma(\delta \uparrow \gamma) = (\delta \uparrow \gamma) \uparrow (\delta \uparrow \gamma)$, both of which are identities, it is clear that $\gamma(\delta \uparrow \gamma)$ is a $\precsim$-upper bound of γ and δ. Suppose ξ is any $\precsim$-upper bound of γ and δ. Then

$$\gamma(\delta \uparrow \gamma) \uparrow \xi = (\gamma \uparrow \xi)((\delta \uparrow \gamma) \uparrow (\xi \uparrow \gamma)) = (\gamma \uparrow \xi)((\delta \uparrow \xi) \uparrow (\gamma \uparrow \xi)),$$

which is an identity, so $\gamma(\delta \uparrow \gamma) \precsim \xi$. ∎

Theorem 3 *Suppose $A = (E, Q, T)$ is a trace automaton, and $q \in Q$. Then the set of permutation equivalence classes of computation sequences from state q, equipped with the partial order induced by $\underset{\sim}{\sqsubset}$, is a domain, whose finite elements are exactly the equivalence classes of finite computation sequences.*

Proof – The set of finite computation sequences from state q is countable, and from Lemma 2.3 we know that whenever γ and δ have an upper bound with respect to $\underset{\sim}{\sqsubset}$, then they have a supremum with respect to $\underset{\sim}{\sqsubset}$. Then the ideal completion $\mathcal{I}$ of the preorder $\underset{\sim}{\sqsubset}$ is a domain whose finite elements are exactly the principal ideals. Let h be the map that takes each $\underset{\sim}{\sqsubset}$-equivalence class $[\gamma]$ to the set $h([\gamma]) = \{\delta : \delta \text{ finite}, \delta \underset{\sim}{\sqsubset} \gamma\}$. The set $h([\gamma])$ is obviously nonempty and downward-closed. It is also directed, because if $\delta \in h([\gamma])$ and $\delta' \in h([\gamma])$, then δ and δ' have a $\underset{\sim}{\sqsubset}$-supremum, which must also be in $h([\gamma])$. Thus, $h([\gamma])$ is an ideal of $\underset{\sim}{\sqsubset}$.

We claim that the map h is an order-isomorphism, from the partially ordered set of permutation equivalence classes of computation sequences from state q, to $\mathcal{I}$. Since h takes each equivalence class $[\gamma]$ with γ finite to the principal ideal generated by γ, we will then have the desired result. Obviously h satisfies $h([\gamma]) \subseteq h([\delta])$ iff $\gamma \underset{\sim}{\sqsubset} \delta$. Note that h is injective, because if $[\gamma] \neq [\delta]$ then either γ has a finite prefix γ' such that $\gamma' \notin h([\delta])$, or else δ has a finite prefix δ' such that $\delta' \notin h([\gamma])$. To complete the proof, we must show that h is also surjective; that is, every $\underset{\sim}{\sqsubset}$-ideal Γ of the set of finite computation sequences is $h([\gamma])$ for some computation sequence γ.

Suppose $\Gamma \in \mathcal{I}$. We first observe that Γ is at most countable (because the set of all finite computation sequences is countable), hence has an enumeration (perhaps with repetition) $\delta_0, \delta_1, \ldots$. Next, we inductively construct a sequence $\gamma_0 \leq \gamma_1 \leq \ldots$ of elements of Γ, forming a chain under the prefix ordering, such that $\delta_k \underset{\sim}{\sqsubset} \gamma_{k+1}$ for all $k \geq 0$. For the basis step, let $\gamma_0 = \mathrm{id}_q$, which is in Γ because Γ is an ideal. For the induction step, suppose $\gamma_k \in \Gamma$ has been defined for some $k \geq 0$. Since $\delta_k, \gamma_k \in \Gamma$, and Γ is directed, it follows by Lemma 2.3 that δ_k and γ_k are consistent. Define $\gamma_{k+1} = \gamma_k(\delta_k \uparrow \gamma_k)$. Clearly, γ_k is a prefix of γ_{k+1}. Since γ_{k+1} is a $\underset{\sim}{\sqsubset}$-supremum of $\{\gamma_k, \delta_k\} \subseteq \Gamma$, and since the ideal Γ is closed under suprema of finite subsets, it follows that $\gamma_{k+1} \in \Gamma$. Also, $\delta_k \underset{\sim}{\sqsubset} \gamma_{k+1}$, since $\delta_k \uparrow \gamma_{k+1} = (\delta_k \uparrow \gamma_k) \uparrow (\delta_k \uparrow \gamma_k) = \mathrm{id}$.

Let γ be the supremum of the chain $\gamma_0 \leq \gamma_1 \leq \ldots$ with respect to the prefix ordering. We claim that $h([\gamma]) = \Gamma$. Clearly, if $\delta \in \Gamma$, then $\delta = \delta_k$ for some $k \geq 0$, hence $\delta \underset{\sim}{\sqsubset} \gamma_{k+1}$. This shows $\Gamma \subseteq h([\gamma])$. Conversely, if δ is a finite computation sequence with $\delta \underset{\sim}{\sqsubset} \gamma$, then $\delta \underset{\sim}{\sqsubset} \xi$ for some finite prefix ξ of γ. But this means $\delta \underset{\sim}{\sqsubset} \gamma_k$ for some $k \geq 0$, hence $h([\gamma]) \subseteq \Gamma$ because $\gamma_k \in \Gamma$ and Γ is an ideal. ∎

To proceed further, we need some additional information about the structure of the domain of traces $\bar{E}$ generated by a concurrent alphabet E. Define a *trace domain* to be a structure $(D, \sqsubseteq, \perp, \cdot)$, where $(D, \sqsubseteq, \perp)$ is a finitary domain with least element $\perp$, and if D° denotes the set of finite elements of D, then $(D^\circ, \cdot, \perp)$ is a monoid. In addition, the following are required to hold:

1. For all $x, y \in D^\circ$, we have $x \sqsubseteq y$ iff there exists $z \in D^\circ$ with $xz = y$.

2. For all $x, y, z \in D^\circ$, if $xy = xz$ then $y = z$.

3. For all distinct atoms $a, b \in D$, we have a, b consistent iff $ab = ba$, and then $a \sqcup b = ab = ba$.

On any trace domain D, we may define a partial binary operation $\setminus$ on D^{o}, such that $x \setminus y$ is defined iff x and y are consistent, and if so, then $x \setminus y$ is the unique z such that $yz = x \sqcup y$. Obviously the domain of definition of $\setminus$ is symmetric, and $x = y$ iff $x \setminus y = \epsilon = y \setminus x$.

Lemma 2.4 *Suppose $(D, \sqsubseteq, \perp, \cdot)$ is a trace domain. Then the following identities hold for finite x, y, z, whenever either side is defined:*

1. $x(y \sqcup z) = xy \sqcup xz$.

2. $z \setminus xy = (z \setminus x) \setminus y$.

3. $xy \setminus z = (x \setminus z)(y \setminus (z \setminus x))$.

Proof – Omitted. ∎

We will prove the following lemma in Section 3.5 below.

Lemma 2.5 *Suppose E is a concurrent alphabet. Then $(\bar{E}, \sqsubseteq, \epsilon, \cdot)$ is a trace domain. Conversely, a structure $(D, \sqsubseteq, \perp, \cdot)$ is a trace domain iff it is isomorphic to $(\bar{E}, \sqsubseteq, \epsilon, \cdot)$ for some concurrent alphabet E.*

Lemma 2.6 *Suppose γ and δ are coinitial finite computation sequences of a trace automaton. Then γ and δ are consistent iff $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\delta)$ are consistent, in which case $\mathrm{tr}(\gamma \uparrow \delta) = \mathrm{tr}(\gamma) \setminus \mathrm{tr}(\delta)$.*

Proof – The proof is by induction on the pair of lengths $(|\gamma|, |\delta|)$. For the special cases: $|\gamma| = 0$ or $|\delta| = 0$, the result is trivial. For the basis case of $|\gamma| = |\delta| = 1$ we have that γ is a single transition $t : q \xrightarrow{a} r$ and δ is a single transition $u : q \xrightarrow{b} s$, so $\mathrm{tr}(\gamma) = [a]$ and $\mathrm{tr}(\delta) = [b]$. Then γ and δ are consistent iff one of the following cases occurs:

1. $a = b$.

2. $a \neq b$ and $a \|_E b$.

By Lemma 2.5, traces $[a]$ and $[b]$ are consistent iff one of these two cases occurs. In case (1), traces $[a]$ and $[b]$ are equal, and $\mathrm{tr}(\gamma \uparrow \delta) = \epsilon = \mathrm{tr}(\gamma) \setminus \mathrm{tr}(\delta)$. In case (2), $[a] \setminus [b] = [a]$, hence $\mathrm{tr}(\gamma \uparrow \delta) = [a] = \mathrm{tr}(\gamma) \setminus \mathrm{tr}(\delta)$.

For the induction step, suppose $|\gamma| > 1$ and $|\delta| \geq 1$. (We omit the case $|\gamma| \geq 1$ and $|\delta| > 1$, which is symmetric.) Then $\gamma = \xi\zeta$, where $|\xi| \geq 1$ and $|\zeta| \geq 1$. Suppose $\mathrm{tr}(\xi) = [x]$, $\mathrm{tr}(\zeta) = [y]$, and $\mathrm{tr}(\delta) = [z]$. By Lemma 2.1, γ and δ are consistent iff ξ and δ are consistent and also $\delta \uparrow \xi$ and ζ are consistent. If γ and δ are consistent, then

$$\begin{aligned} \gamma \uparrow \delta &= (\xi \uparrow \delta)(\zeta \uparrow (\delta \uparrow \xi)) \\ \delta \uparrow \gamma &= (\delta \uparrow \xi) \uparrow \zeta. \end{aligned}$$

Applying the induction hypothesis and Lemma 2.4, we see that

$$\begin{aligned}
\mathrm{tr}(\gamma \uparrow \delta) &= (\mathrm{tr}(\xi) \uparrow \mathrm{tr}(\delta))(\mathrm{tr}(\zeta) \uparrow (\mathrm{tr}(\delta) \uparrow \mathrm{tr}(\xi))) = [x][y] \setminus [z] = \mathrm{tr}(\gamma) \setminus \mathrm{tr}(\delta) \\
\mathrm{tr}(\delta \uparrow \gamma) &= (\mathrm{tr}(\delta) \uparrow \mathrm{tr}(\xi)) \uparrow \mathrm{tr}(\zeta) = [z] \setminus [x][y] = \mathrm{tr}(\delta) \setminus \mathrm{tr}(\gamma).
\end{aligned}$$

Conversely, suppose $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\delta)$ are consistent. Since then $\mathrm{tr}(\xi)$ and $\mathrm{tr}(\delta)$ are consistent, by the induction hypothesis we know that ξ and δ are consistent, $\mathrm{tr}(\delta \uparrow \xi) = \mathrm{tr}(\delta) \setminus \mathrm{tr}(\xi)$, and $\mathrm{tr}(\xi \uparrow \delta) = \mathrm{tr}(\xi) \setminus \mathrm{tr}(\delta)$. Now, since $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\delta)$ are consistent, and $\mathrm{tr}(\xi) \sqsubseteq \mathrm{tr}(\gamma)$, it follows that $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\xi) \sqcup \mathrm{tr}(\delta)$ are consistent. Moreover, $\mathrm{tr}(\gamma) = \mathrm{tr}(\xi)\mathrm{tr}(\zeta)$ and $\mathrm{tr}(\delta) \sqcup \mathrm{tr}(\xi) = \mathrm{tr}(\xi)(\mathrm{tr}(\delta) \setminus \mathrm{tr}(\xi))$, so it follows that $\mathrm{tr}(\zeta)$ and $\mathrm{tr}(\delta) \setminus \mathrm{tr}(\xi)$ are consistent. Applying the induction hypothesis once again shows that ζ and $\delta \setminus \xi$ are consistent. Since ξ and δ are consistent, and $\delta \uparrow \xi$ and ζ are consistent, it follows that γ and δ are consistent, as was to be shown. ∎

A *subdomain* of a domain D is a subset U of D, which is a domain under the restriction of the ordering on D, and is such that the inclusion of U in D is strict and continuous. A subdomain U of D is *normal* if for all $d \in D$, the set $\{e \in U : e \sqsubseteq d\}$ is directed.

Lemma 2.7 *Domain D' is isomorphic to a normal subdomain of D iff there exists a strict, additive, continuous injection $f : D' \to D$ that also reflects consistency.*

Proof – Omitted. ∎

An *interval* in a domain D is a pair (d, d') of elements of D, with $d \sqsubseteq d'$. A *prime* interval is an interval (d, d') such that $d \sqsubset d'$, and there exists no $d'' \in D$ with $d \sqsubset d'' \sqsubset d'$.

Theorem 4 *Suppose D is the domain of permutation equivalence classes of initial computation sequences of a trace automaton with start state $A = (E, Q, T, *)$. Then D is isomorphic to a normal subdomain U of $\bar{E}$, such that the inclusion of U in $\bar{E}$ preserves prime intervals. Conversely, if U is a normal subdomain of $\bar{E}$, such that the inclusion of U in $\bar{E}$ preserves prime intervals, then U is isomorphic to the domain of permutation equivalence classes of initial computation sequences of a trace automaton with start state.*

Proof – Suppose domain D is the domain of permutation equivalence classes $[\gamma]$ of initial computation sequences γ of a trace automaton with start state $A = (E, Q, T, *)$. Each computation sequence γ of A determines a trace $\mathrm{tr}(\gamma) \in \bar{E}$. By Lemma 2.6, if $\gamma \precsim \delta$, then $\mathrm{tr}(\gamma) \setminus \mathrm{tr}(\delta) = \epsilon$, so $\mathrm{tr}(\gamma) \sqsubseteq \mathrm{tr}(\delta)$. It follows that the mapping tr from computation sequences to traces induces a monotone injection (which we also denote by tr) from D to $\bar{E}$. Clearly, the map tr is strict. It also preserves prime intervals, because an interval $([\gamma], [\delta])$ in D is prime iff $\delta \sim \gamma t$ for some nonidentity transition t, and an interval $([x], [y])$ in $\bar{E}$ is prime iff $y \sim xe$ for some $e \in E$.

To see that tr is continuous, it suffices to show that it preserves suprema of $\precsim$-chains. Suppose $\gamma_0 \precsim \gamma_1 \precsim \ldots$ is a chain with supremum γ. By a dovetailing argument using ω-algebraicity and consistent completeness, we may construct a prefix-ordered chain of finite computation sequences $\delta_0 \leq \delta_1 \leq \ldots$ with the following properties:

1. For all $j \geq 0$, there exists $k \geq 0$ such that $\delta_j \precsim \gamma_k$.

2. For all $k \geq 0$, and all finite $\xi \precsim \gamma_k$, there exists $j \geq 0$ such that $\xi \precsim \delta_j$.

It follows that $\bigsqcup_k \mathrm{tr}(\delta_k) = \bigsqcup_k \mathrm{tr}(\gamma_k)$, and if δ is the $\leq$-supremum of $\delta_0 \leq \delta_1 \leq \ldots$, then $\delta \sim \gamma$. Since it is obvious from the definition that tr is continuous with respect to the prefix ordering $\leq$, we may then conclude that $\mathrm{tr}(\gamma) = \mathrm{tr}(\delta) = \bigsqcup_k \mathrm{tr}(\delta_k) = \bigsqcup_k \mathrm{tr}(\gamma_k)$.

To see that tr is additive, suppose γ and δ are consistent finite computation sequences. Then $\xi = \gamma(\delta \uparrow \gamma)$ is a $\precsim$-supremum of γ and δ. Moreover, $\mathrm{tr}(\xi) = \mathrm{tr}(\gamma)(\mathrm{tr}(\delta) \setminus \mathrm{tr}(\gamma)) = \mathrm{tr}(\gamma) \sqcup \mathrm{tr}(\delta)$ by Lemmas 2.4 and 2.5. Finally, if $\mathrm{tr}(\gamma)$ and $\mathrm{tr}(\delta)$ are consistent, then γ and δ are consistent by Lemma 2.6, so tr reflects consistency. Since the map tr is a strict, additive, continuous injection that reflects consistency, it is a isomorphism from D to a normal subdomain of $\bar{E}$ by Lemma 2.7.

Conversely, suppose U is a normal subdomain of $\bar{E}$ for some concurrent alphabet E, such that the inclusion of U in $\bar{E}$ preserves prime intervals. Let $A = (E, U^\circ, T, \epsilon)$, where U° is the set of finite elements of U, and where T contains id_ϵ all identity transitions $\mathrm{id}_{[x]}$ with $[x] \in U^\circ$, and all transitions $[x] \xrightarrow{a} [xa]$ where both $[x]$ and $[xa] \in U^\circ$. It is easy to check that A satisfies the conditions for an automaton. If γ is an initial computation sequence of A, then $\mathrm{tr}(\delta)$ is a finite element of U for each finite prefix δ of γ, so $\mathrm{tr}(\gamma) \in U$ by continuity of tr. Conversely, if $[x] \in U$, then we may choose a chain $[x_0] \sqsubseteq [x_1] \sqsubseteq \ldots$ of finite elements of U such that $\bigsqcup_k [x_k] = [x]$. Since the inclusion of U in $\bar{E}$ preserves prime intervals, and the domain $\bar{E}$ is finitary, we may assume without loss of generality that for each $k \geq 0$, either $x_{k+1} = x_k$ or else $x_{k+1} = x_k a$ for some $a \in E$. It is then a simple matter to construct a corresponding initial computation sequence γ of A, such that $\mathrm{tr}(\gamma) = [x]$. ∎

It can be shown (see [14]), that a domain D is isomorphic to a normal subdomain U of a domain of traces $\bar{E}$, where the inclusion of U in $\bar{E}$ preserves prime intervals, if and only if D is an *event domain* [6, 18]. Event domains, in turn, are those that are isomorphic to the domains of "configurations" of an "event structure" determined by an enabling relation and a binary conflict relation. Thus, trace automata generate the same class of domains as event structures.

3 Concurrent Transition Systems

The notion of the residual of one computation sequence after another was crucial in the results of the previous section. The properties of the residual operation that were used in the proofs can be axiomatized, and the result is *concurrent transition systems* [15, 16]. As we have shown in these papers and in [12, 17], many interesting properties of automata can be established from these axioms. An advantage of the more abstract, axiomatic formulation is that there is an obvious way in which concurrent transition systems can be made into a category, which we call **CTS**. The category **CTS** has a surprisingly rich structure, and categorical constructions in it and related functor categories correspond directly to many natural constructions we wish to perform on automata, including the extraction of their computational behavior. After investigating the properties of **CTS**, we shall see that a certain mapping from trace automata to **CTS** suggests an interesting way to make trace automata into a category.

An *abstract automaton* is a structure $A = (Q, T, \mathrm{dom}, \mathrm{cod}, \mathrm{id})$, where

- Q is a set of *states*.
- T is a set of *transitions*.
- $\mathrm{dom}, \mathrm{cod} : T \to Q$ are functions that map each transition to its *domain* and *codomain*, respectively.
- $\mathrm{id} : Q \to T$ maps each state q in Q to a distinguished *identity transition* $\mathrm{id}_q \in T$, such that $\mathrm{dom}(\mathrm{id}_q) = \mathrm{cod}(\mathrm{id}_q) = q$.

Let $\mathrm{Coin}(A)$ denote the set of all coinitial pairs of transitions of A. A *residual operation* on an automaton $A = (Q, T, \mathrm{dom}, \mathrm{cod}, \mathrm{id})$ is a partial function $\uparrow : \mathrm{Coin}(A) \to T$, such that the following conditions hold (we write $t \uparrow u$ instead of $\uparrow (t, u)$, and we read it as "*t after u*"):

1. If $t \uparrow u$ is defined, then so is $u \uparrow t$, and then $\mathrm{dom}(t \uparrow u) = \mathrm{cod}(u)$, $\mathrm{dom}(u \uparrow t) = \mathrm{cod}(t)$, and $\mathrm{cod}(t \uparrow u) = \mathrm{cod}(u \uparrow t)$.
2. For all $t : q \to r$ in T, *(a)* $\mathrm{id}_q \uparrow t = \mathrm{id}_r$; *(b)* $t \uparrow \mathrm{id}_q = t$; and *(c)* $t \uparrow t = \mathrm{id}_r$.
3. For all coinitial $t, u, v \in T$, $(v \uparrow t) \uparrow (u \uparrow t) = (v \uparrow u) \uparrow (t \uparrow u)$, whenever either side is defined.

A residual operation is *extensional* if it satisfies the additional condition:

4. For all coinitial transitions t, u, if $t \uparrow u$ and $u \uparrow t$ are both identities, then $t = u$.

A *concurrent transition system* (CTS) is a pair $C = (A, \uparrow)$, where A is an abstract automaton and $\uparrow$ is an extensional residual operation on A. If $\uparrow$ is a residual operation, but not necessarily extensional, then C is called a *pre-CTS*. Coinitial transitions t, u of a pre-CTS are called *consistent* if $t \uparrow u$ is defined (equivalently, if $u \uparrow t$ is defined), otherwise they are called *conflicting*. We think of consistent transitions as representing actions that might arise in a single concurrent computation, and of conflicting transitions as representing incompatible nondeterministic choices. If t and u are consistent, then we think of $t \uparrow u$ as what remains of the transition t after the transition u has occurred.

The results of the previous section allow us to associate with each trace automaton $A = (E, Q, T)$ a CTS C_A whose states, transitions, and identities are those of A, and whose residual operation is defined as in Section 2.1.2. It is also interesting to observe that a concurrent alphabet E can be viewed as the set of nonidentity transitions of a one-state CTS, such that $a, b \in E$ are consistent exactly when either $a = b$ or $a \|_E b$, and in the latter case $a \uparrow b = a$ and $b \uparrow a = b$. We shall put this observation to good use in Section 3.5.

Suppose $A = (Q, T, \mathrm{dom}, \mathrm{cod}, \mathrm{id})$ and $A' = (Q', T', \mathrm{dom}', \mathrm{cod}', \mathrm{id}')$ are automata. Then a *morphism* from A to A' is a pair of maps $\rho = (\rho_{\mathrm{o}}, \rho_{\mathrm{a}})$, where $\rho_{\mathrm{o}} : Q \to Q'$ and $\rho_{\mathrm{a}} : T \to T'$, such that $\mathrm{dom}' \circ \rho_{\mathrm{a}} = \rho_{\mathrm{o}} \circ \mathrm{dom}$, $\mathrm{cod}' \circ \rho_{\mathrm{a}} = \rho_{\mathrm{o}} \circ \mathrm{cod}$, and $\mathrm{id}' \circ \rho_{\mathrm{o}} = \rho_{\mathrm{a}} \circ \mathrm{id}$. In the sequel, we will drop the notational distinction between ρ_{o} and ρ_{a}, writing simply ρ in both cases. Let **Auto** denote the category of automata and their morphisms.

A *morphism* from a pre-CTS $B = (A, \uparrow)$ to a pre-CTS $B' = (A', \uparrow')$ is a morphism $\rho : A \to A'$ of the underlying automata, with the following additional property:

- If t, u are consistent transitions of B, then $\rho(t)$ and $\rho(u)$ are consistent transitions of B', and $\rho(t \uparrow u) = \rho(t) \uparrow' \rho(u)$.

The class of all pre-CTS's forms a category **PCTS**, when equipped with the CTS-morphisms as arrows. Let **CTS** denote the full subcategory of **PCTS** whose objects are the CTS's.

The following result shows that there is a universal way to obtain a CTS from a pre-CTS.

Lemma 3.1 *Suppose B is a pre-CTS. Then there exists a CTS $B^\flat$, and a morphism $\flat_B : B \to B^\flat$, with the following property: if C is any CTS equipped with a morphism $\rho : B \to C$, then there exists a unique morphism $\rho^\flat : B^\flat \to C$ with $\rho = \rho^\flat \circ \flat_B$.*

Proof – Suppose $B = (A, \uparrow)$. Define a binary relation $\sim$ on transitions of B by: $t \sim u$ iff $t \uparrow u = \mathrm{id}$ and $u \uparrow t = \mathrm{id}$. Let $A^\flat$ be the automaton whose states are those of A, but whose transitions are $\sim$-equivalence classes $[t]$ of transitions of A, with domain, codomain, and identities defined in the obvious way. Define $\uparrow^\flat$ by: $[t] \uparrow^\flat [u] = [t \uparrow u]$, and let $B^\flat = (A^\flat, \uparrow^\flat)$. Define $\flat_B : B \to B^\flat$ to take each transition t of A to the corresponding $\sim$-equivalence class $[t]$, which is a transition of $A^\flat$. One may now verify that $B^\flat$ and $\flat_B$ are well-defined and have the required properties. ∎

Corollary 3.1 **CTS** *is a reflective subcategory of* **PCTS**.

Proof – Lemma 3.1 shows that the inclusion of **CTS** in **PCTS** has a left adjoint, whose object map takes B to $B^\flat$. ∎

There is also a universal way to lift an automaton to a CTS.

Lemma 3.2 *Suppose A is an automaton. Then there exists a CTS $A^\sharp = (A, \uparrow)$ with the following property: Given a CTS $C' = (A', \uparrow')$, every morphism $\rho : A \to A'$ in* **Auto** *is also a morphism $\rho : A^\sharp \to C'$ in* **CTS**.

Proof – Given an automaton A, let $t \uparrow u$ be defined for coinitial transitions t, u of A iff either $t = u$ or else either t or u is an identity. If $t = u$ or $t = \mathrm{id}$, then $t \uparrow u = \mathrm{id}$, and if u is an identity, then $t \uparrow u = t$. Let $A^\sharp = (A, \uparrow)$, then it is easy to see that $A^\sharp$ has the required property. ∎

Corollary 3.2 **Auto** *is isomorphic to a coreflective subcategory of* **CTS**.

Proof – Lemma 3.2 shows that the forgetful functor Auto : **CTS** $\to$ **Auto** has a left adjoint, whose object map takes A to $A^\sharp$, and which is full and faithful. ∎

3.1 Basic Consequences of the CTS Axioms

Define a relation $\preceq$ on the transitions of a CTS by: $t \preceq u$ iff t, u are coinitial and $t \uparrow u = \mathrm{id}$. We say that a transition v is a *join* of the coinitial transitions t, u if $t \preceq v$, $u \preceq v$, $v \uparrow t = u \uparrow t$, and $v \uparrow u = t \uparrow u$. If t and u are composable, then we say that a transition v is a *composite* of t, u if $t \uparrow v = \mathrm{id}$ and $v \uparrow t = u$.

The following results are proved in [16]. The reader may enjoy working out the proofs, since they are good examples of how to work with residuals.

Lemma 3.3 *Suppose C is a CTS.*

1. *The relation $\preceq$ is a partial order.*
2. *If a composite of t and u exists, then it is unique. (We denote it by tu.)*
3. *Suppose t and v are coinitial, and the composite tu of t and u exists.*
   (a) $v \uparrow tu = (v \uparrow t) \uparrow u$.
   (b) $tu \uparrow v = (t \uparrow v)(u \uparrow (v \uparrow t))$.
4. *Composition obeys the following laws:*
   (a) *For all t, $t = \mathrm{id}\, t = t\, \mathrm{id}$.*
   (b) i. *If tu and $(tu)v$ exist, then uv and $t(uv)$ exist, and $(tu)v = t(uv)$.*
   ii. *If tu, uv, and $t(uv)$ exist, then $(tu)v$ exists and $(tu)v = t(uv)$.*
   (c) *If tu and tv exist, and $tu = tv$, then $u = v$.*
5. *A transition v is a join of t and u iff $v = t(u \uparrow t)$. Hence a join of t and u, if it exists, is unique. (We denote it by $t \vee u$.) Moreover, if $t \vee u$ exists, then it is the least upper bound of t and u under $\preceq$.*

Lemma 3.4 *Suppose $\rho : C \to C'$ is a morphism. Then*

1. *$\rho(tu) = \rho(t)\rho(u)$ whenever tu exists.*
2. *$\rho(t \vee u) = \rho(t) \vee \rho(u)$ whenever $t \vee u$ exists.*

3.2 Completion of a CTS

A CTS is called *complete* if every composable pair of transitions has a composite. Complete CTS's play a fundamental role in the process of extracting the computational behavior of a CTS. In particular, each CTS C freely generates a *completion C^**, whose states are the same as those of C, and whose transitions are certain equivalence classes of finite sequences of transitions of C. In case C is the CTS C_A corresponding to a trace automaton A, then the transitions of C_A^* are precisely the permutation equivalence classes of finite computation sequences of A.

Lemma 3.5 *Suppose C is a CTS. Then there exists a complete CTS C^*, and a morphism $\natural_C : C \to C^*$, with the following property: if C' is any complete CTS equipped with a morphism $\rho : C \to C'$, then there exists a unique morphism $\rho^* : C^* \to C'$ such that $\rho = \rho^* \circ \natural_C$.*

Proof – Suppose $C = (A, \uparrow)$ is a CTS. Let A^* be the automaton whose states are those of A, and whose transitions are the finite computation sequences of A, with the computation sequences of length 0 as the identity transitions. If we identify each nonidentity transition of A with the corresponding computation sequence of length 1, then the residual operation $\uparrow$ on A extends uniquely to a residual operation $\uparrow^*$ on A^*, such that the following hold whenever either side is defined:

$$\xi \uparrow^* \gamma\delta = (\xi \uparrow^* \gamma) \uparrow^* \delta, \qquad \gamma\delta \uparrow^* \xi = (\gamma \uparrow^* \xi)(\delta \uparrow^* (\xi \uparrow^* \gamma)).$$

Then $B = (A^*, \uparrow^*)$ is a pre-CTS. Moreover, the map $\mu : C \to B$ that takes each state of C to the same state of B and each nonidentity transition of C to the corresponding computation sequence of length 1, is a morphism. Define $C^* = B^\flat$ and define $\natural_C : C \to C^*$ by: $\natural_C = \flat_B \circ \mu$. One may now verify that C^* and $\natural_C$ have the stated properties. ∎

Let **CCTS** denote the full subcategory of **CTS** having the complete CTS's as objects.

Corollary 3.3 **CCTS** *is a reflective subcategory of* **CTS**.

Proof – Lemma 3.5 shows that the inclusion of **CCTS** in **CTS** has a left adjoint, whose object map takes C to C^*. ∎

Theorem 5 *If C_A is the CTS associated with a trace automaton A, then up to isomorphism, C_A^* is the CTS whose states are the states of A and whose transitions are the permutation equivalence classes of finite computation sequences of A, with residual given by Lemma 2.1.*

Proof – By the construction in Lemma 3.5, C_A^* has as states the states of A and as transitions the $\sim$-equivalence classes of finite computation sequences of A, where $\gamma \sim \delta$ holds iff $\gamma \uparrow^* \delta = \mathrm{id}$ and $\delta \uparrow^* \gamma = \mathrm{id}$. By Theorem 2, $\sim$ is nothing more than permutation equivalence. ∎

In later sections, we need a characterization of the complete CTS's that was proved in [16]. Define a *computation category* to be a small category C with the following properties:

1. Every arrow is an epimorphism.

2. The only isomorphisms are identities.

3. C has *bounded pushouts*: whenever t, u are coinitial arrows of C such that $tv = uw$ for some arrows v, w, then t and u have a pushout.

For the statement of the next result, we observe that a category is nothing more than a pair $(A, \cdot)$, where A is an automaton and $\cdot$ is an associative composition on the transitions of A, having the identities of A as units.

Lemma 3.6 *Suppose $C = (A, \uparrow)$ is a complete CTS, and let $\cdot$ denote the composition operation of C. Then $C' = (A, \cdot)$ is a computation category. Conversely, suppose $C' = (A, \cdot)$ is a computation category. For coinitial arrows t, u of A, let $t \uparrow u$ be defined iff $tv = uw$ for some arrows v, w, in which case let $t \uparrow u$ be the side opposite t in a pushout square with t, u as its base. Then $C = (A, \uparrow)$ is a complete CTS, whose composition operation coincides with that of C.*

Proof – Omitted. ∎

3.3 Computation Diagrams

In this section we generalize to CTS's the notions of "computation tree" and "computation" for ordinary transition systems. Given a CTS C, the set of all transitions of its completion C^* from a designated state $*$, forms the set of states of another complete CTS D, which we call the *complete computation diagram* of C with respect to $*$. Transitions of D represent prefix relationships between concurrent computations; accordingly, there is at most one transition between any two states of D.

Formally, a *pointed CTS* is a pair $(C, *)$, where C is a CTS, and $*$ is a distinguished state of C, called the *start state.* Let $\mathbf{CTS}_*$ (resp. $\mathbf{CCTS}_*$) denote the category of pointed CTS's (resp. pointed, complete CTS's) and morphisms that preserve the start state.

A *complete computation diagram* is a pointed CTS $(D, \perp)$, such that D is complete, and for each state q of D, there is a unique transition $0_q : \perp \to q$ in D. The $\preceq$ partial order on transitions of D determines a corresponding ordering $\preceq$ on states of D, defined by: $q \preceq r$ iff $0_q \preceq 0_r$. Then the set of states of D, partially ordered by $\preceq$, has $\perp$ as a least element, and by Lemma 3.6 has the property that any pair of states with an upper bound, has a least upper bound.

There is a universal way to obtain a complete computation diagram from a pointed, complete CTS.

Lemma 3.7 *Suppose $(C, *)$ is a pointed, complete CTS. Then there exists a complete computation diagram $\mathrm{CDiag}(C, *) = (D, \perp)$, and a $\mathbf{CTS}_*$-morphism*

$$\varepsilon : \mathrm{CDiag}(C, *) \to (C, *),$$

with the following property: if $(D', \perp')$ is any other complete computation diagram, equipped with a morphism $\rho : (D', \perp') \to (C, *)$, then there exists a unique $\mathbf{CTS}_*$-morphism $\rho^\dagger : (D', \perp') \to \mathrm{CDiag}(C, *)$ such that $\rho = \varepsilon \circ \rho^\dagger$.

Proof – Given $(C, *)$, let D be the complete CTS whose states are all transitions t of C such that $\mathrm{dom}(t) = *$, and in which there is a unique transition from t to u iff $t \preceq u$. Let $\perp = \mathrm{id}_*$. Let $\varepsilon : (D, \perp) \to (C, *)$ take each state t of D to the state $\mathrm{cod}(t)$

of C, and each transition from t to u in D to the transition $u \uparrow t$ in C. One may now easily check that $(D, \perp)$ and ε have the required properties. ∎

Let **CDiag** denote the full subcategory of $\mathbf{CTS}_*$ whose objects are the complete computation diagrams.

Theorem 6 **CDiag** *is coreflective in* $\mathbf{CCTS}_*$.

Proof – Lemma 3.7 shows that the inclusion of **CDiag** in $\mathbf{CCTS}_*$ has a right adjoint, whose object map takes $(C, *)$ to $\mathrm{CDiag}(C, *)$. ∎

Theorem 7 *Suppose* $(C_A, *)$ *is the pointed CTS determined by a trace automaton* A *with start state, and let* $(D, \perp) = \mathrm{CDiag}(C_A, *)$ *be its complete computation diagram. Then up to isomorphism,* D *is the poset of finite initial concurrent computations of* A.

Proof – Obvious from Theorem 5 and the construction of $\mathrm{CDiag}(C_A, *)$ given in the proof of Lemma 3.7. ∎

Noting that the domain of all computations of A may be obtained by ideal completion of the poset of finite computations, we make the following general definition:

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

3.4 The Category CTS

The category **CTS** has a great deal of structure making it suitable for use in constructing models of concurrent systems.

Theorem 8 *The category* **CTS**:

1. *has equalizers and small products, hence all small limits.*
2. *has small coproducts.*
3. *is cartesian closed.*
4. *has small filtered colimits.*

Proof – The proofs of (1) and (2) use the obvious constructions. Assertion (3) is proved in [16].

To show (4), let D be a small filtered category, and let $L : D \to \mathbf{CTS}$ be a functor. Our objective is to construct a colimit of L. Let V denote the set of objects of D, and let E denote the set of its arrows. For each $i \in V$, let $C_i = (A_i, \uparrow_i)$ denote the CTS Li, where $A_i = (Q_i, T_i, \mathrm{dom}_i, \mathrm{cod}_i, \mathrm{id}_i)$. Define T to be the disjoint union $\coprod_{i \in V} T_i$. Let the relation $\sim$ on T be defined as follows: $t \sim u$ iff there exist arrows $f : i \to k$ and $g : j \to k$ in D such that $t \in T_i$, $u \in T_j$, and $Lf(t) = Lg(u)$. One may now show that $\sim$ is an equivalence relation, and then construct a CTS C having the equivalence classes of $\sim$ as its transitions. The CTS C is the base of a colimiting cone over L. ∎

3.5 Application to Trace Theory

In this section, we show how the theory of CTS's can be used to obtain results in trace theory. Typically, proofs in trace theory make use of the representation of traces by their "dependency graphs" [1, 11]. In contrast, proofs using CTS theory involve residuals. One of our goals is to prove Lemma 2.5, which we have already used in the proof of Theorem 4.

We first establish a correspondence between concurrent alphabets and certain CTS's. This correspondence extends to yield a correspondence between free partially commutative monoids and certain complete CTS's.

Lemma 3.8 *Suppose E is a concurrent alphabet. Then E is the set of nonidentity transitions of a one-state CTS C_E, in which $a \uparrow b$ is defined for $a, b \in E$ precisely when $a \|_E b$, and in that case $a \uparrow b = a$. Conversely, suppose C is a one-state CTS in which $a \uparrow b = a$ whenever a and b are consistent, distinct, nonidentity transitions. Let E_C be the concurrent alphabet whose elements are the nonidentity transitions of C, with $a \|_{E_C} b$ defined to hold iff a and b are consistent and distinct. Then $E_{C_E} = E$ and $C_{E_C} \simeq C$.*

Proof – Straightforward. ∎

Lemma 3.9 *Suppose C is a one-state complete CTS. Then $x \uparrow y$ is defined iff x and y have an upper bound with respect to the ordering $\preceq$. Moreover, the transitions of C, equipped with the identity and composition of C, form a monoid* $\mathrm{Mon}(C)$ *with the following properties:*

1. *For all $x, y, z \in \mathrm{Mon}(C)$, if $xy = xz$ then $y = z$.*
2. *Define $x \sqsubseteq y$ iff $\exists z(xz = y)$. Then $\sqsubseteq$ is a consistently complete partial order with the identity transition of C as a least element.*

Proof – Immediate from Lemma 3.6. ∎

From Lemma 3.9, it follows that the residual operation on C can be recovered from the monoid $\mathrm{Mon}(C)$, because $x \uparrow y$ is defined iff x and y have an upper bound with respect to $\sqsubseteq$ (which is the same relation as $\preceq$), in which case $x \uparrow y$ is the unique z such that $yz = x \sqcup y$.

Lemma 3.10 *Suppose E is a concurrent alphabet. Then* $\mathrm{Mon}(C_E) \simeq E^* /\!\sim$.

Proof – The transitions of C_E^* are permutation equivalence classes $[\gamma]$ of finite computation sequences γ of C_E. Define a map μ from $\mathrm{Mon}(C_E^*)$ to $E^* /\!\sim$ by: $\mu([\gamma]) = \mathrm{tr}(\gamma)$. Note that γ is well-defined, because if $[\gamma] = [\delta]$, then $\mathrm{tr}(\gamma) = \mathrm{tr}(\delta)$ by Theorem 1. Also, μ is a monoid homomorphism, because $\mu(\mathrm{id}) = \epsilon$ and $\mu([\gamma][\delta]) = \mu([\gamma\delta]) = \mathrm{tr}(\gamma\delta) = \mathrm{tr}(\gamma)\mathrm{tr}(\delta) = \mu([\gamma])\mu([\delta])$. The map μ is injective, because if $\mu([\gamma]) = \mu([\delta])$, then $\mathrm{tr}(\gamma) = \mathrm{tr}(\delta)$ hence $[\gamma] = [\delta]$ by Theorem 1. Finally, μ is surjective, because if $[x] \in E^* /\!\sim$, then we may construct a computation sequence γ of C_E, such that the sequence of events appearing in γ is x, hence such that $\mathrm{tr}(\gamma) = [x]$. Since μ is a bijective monoid homomorphism, it is an isomorphism. ∎

We are now equipped to prove Lemma 2.5.

Proof of Lemma 2.5 – ($\Rightarrow$) Suppose E is a concurrent alphabet. By definition of $\bar{E}$, the set $\bar{E}^{\circ}$ of finite elements of $\bar{E}$ is the set of elements of a monoid isomorphic to the free partially commutative monoid $(E^* /\!\sim_E, \cdot, \epsilon)$. By Lemma 3.10, this monoid is isomorphic to the monoid $\mathrm{Mon}(C_E^*)$. Since the residual operation of C_E^* can be recovered from $\mathrm{Mon}(C_E^*)$ it makes sense to use this operation to reason about $\mathrm{Mon}(C_E^*)$, hence about $E^* /\!\sim$.

Now, by Lemma 3.9, the relation $\sqsubseteq$ on $\mathrm{Mon}(C_E^*)$ is a consistently complete partial order with ϵ as a least element. Since the ideal completion of a consistently complete partial order with a least element is a domain, it follows that $\bar{E}$ is a domain. We observe that $\bar{E}$ is finitary, because if $[x]$ is a finite element of $\bar{E}$, then $|\{[y] : [y] \sqsubseteq [x]\}|$ is bounded by the number of prefixes of permutations of x, which is finite.

Property (1) of a trace domain holds for $\bar{E}$ by definition of $\sqsubseteq$. Property (2) of a trace domain is immediate from Lemma 3.9. It remains to verify property (3) of a trace domain. The atoms of $E^* /\!\sim$ are precisely the $\sim$-equivalence classes $[a]$ of elements a of E. Suppose $[a]$ and $[b]$ are distinct atoms. By Lemma 3.9, $[a]$ and $[b]$ are consistent iff $[a] \uparrow [b]$ is defined. But by definition of C_E, $[a] \uparrow [b]$ is defined for distinct $[a], [b]$ iff $a \|_E b$, hence iff $[a][b] = [b][a]$. In that case, $[a] \sqcup [b] = [a]([b] \uparrow [a]) = [a][b] = [b][a]$, by definition of $\uparrow$ on C_E.

($\Leftarrow$) Conversely, suppose $(D, \sqsubseteq, \cdot, \perp)$ is a trace domain. Define a concurrent alphabet E whose elements are the atoms of D, and which has $a \|_E b$ iff $a \neq b$ and $[a][b] = [b][a]$. By property (3) of a trace domain, the monoid homomorphism that takes each element $[a_1 \dots a_n]$ of E^* (with $a_1, \dots, a_n \in E$) to the corresponding finite element $a_1 \dots a_n$ of D respects $\sim$, thus induces a monoid homomorphism from $(E^* /\!\sim) \to D^{\circ}$, and this monoid homomorphism extends uniquely to a continuous map $\mu : \bar{E} \to D$. Property (1) of a trace domain, together with the fact that D is finitary, implies that every element of D° factors via $\cdot$ into a finite sequence of atoms. Thus, μ is surjective.

It remains to be shown that μ is injective. It suffices to show that μ is injective when restricted to $E^* /\!\sim$, since then the injectiveness of μ on all of $\bar{E}$ will follow by algebraicity. It is easy to see that $\mu(x) = \perp$ iff $x = \epsilon$. A straightforward argument by induction on the length of a factorization into atoms shows that $[x] \setminus [y]$ is defined in $\bar{E}$ iff $\mu([x]) \setminus \mu([y])$ is defined in D, and then $\mu([x] \setminus [y]) = \mu([x]) \setminus \mu([y])$. Hence, $\mu([x]) = \mu([y])$ iff $\mu([x]) \setminus \mu([y]) = \perp = \mu([y]) \setminus \mu([x])$, which holds iff $\mu([x] \setminus [y]) = \epsilon = \mu([y] \setminus [x])$, that is, iff $[x] \setminus [y] = \epsilon = [y] \setminus [x]$. Thus, μ is injective, hence is an isomorphism. ∎

The correspondence between concurrent alphabets and one-state CTS's suggests a way to make the class of concurrent alphabets into a category. Formally, if E and F are concurrent alphabets, then define a *strong morphism* from E to F to be an ϵ-preserving map $\mu : (E \cup \{\epsilon\}) \to (F \cup \{\epsilon\})$, such that $a \|_E b$ implies $\mu(a) \neq \mu(b)$. Let **SAlph** denote the category of concurrent alphabets and strong morphisms.

Lemma 3.11 *The map that takes a concurrent alphabet E to the CTS C_E extends* *to an equivalence of* **SAlph** *to the full subcategory of* **CTS** *whose objects are the one-state CTS's C such that $a \uparrow b = a$ whenever a and b are distinct consistent nonidentity transitions of C.*

Proof – Omitted. ∎

The category **SAlph** is not particularly interesting. However, there is another way to map concurrent alphabets to one-state CTS's. This mapping leads to an alternative notion of "weak morphism" of concurrent alphabets. The resulting category **Alph** of concurrent alphabets and weak morphisms is also equivalent to a full subcategory of **CTS**, and is much more interesting than **SAlph**.

Formally, if E is a concurrent alphabet, then call a subset U of E *commuting* if $a \|_E b$ whenever a and b are distinct elements of E. Let $\mathrm{Com}(E)$ denote the set of all finite commuting subsets of E. A *weak morphism* from E to F is a function $\mu : \mathrm{Com}(E) \to \mathrm{Com}(F)$ such that

1. $\mu(\emptyset) = \emptyset$.
2. If $U \cup V \in \mathrm{Com}(E)$, then $\mu(U) \cup \mu(V) \in \mathrm{Com}(F)$, and $\mu(U \setminus V) = \mu(U) \setminus \mu(V)$.

Here the symbol $\setminus$ denotes set difference. Let **Alph** denote the category of concurrent alphabets and weak morphisms.

A CTS C is called *join-complete* if every consistent coinitial pair of transitions of C has a join. The *join-completion* C is the sub-CTS $\widehat{C}$ of C^* whose transitions are precisely those transitions of C^* that can be expressed as a finite join $t_1 \vee \ldots \vee t_n$, where $t_1, \ldots, t_n$ are transitions of C.

Theorem 9 *The map that takes a concurrent alphabet E to the CTS $\widehat{C}_E$ extends to an equivalence of* **Alph** *to the full subcategory of* **CTS** *whose objects are the one-state, join-complete CTS's C with the following properties:*

1. *$t \uparrow u = t$ whenever t and u are distinct, consistent, $\preceq$-minimal nonidentity transitions.*
2. *Every nonidentity transition t of C can be expressed as a join $t_1 \vee \ldots \vee t_n$ of a nonempty set of $\preceq$-minimal nonidentity transitions.*

Proof – Omitted. ∎

We do not have space here to develop in detail the features of the category **Alph**. We merely note that it can be shown that **Alph** has binary products and coproducts, which correspond to intuitively appealing notions of "concurrent product" and "nondeterministic sum," respectively.

3.6 Characterization of Trace CTS's

Notably absent from the CTS definition are any sort of "concreteness" axioms that would have as a consequence, for example, a theorem stating that every transition factors into a finite sequence of $\preceq$-minimal transitions. Part of the reason we have not included such axioms is that we can often do without them, using instead a principle of "computational induction" that arises out of the fact that C^* is freely generated by C. However, another reason we have not given any concreteness axioms is that we are still looking for attractive axioms that are satisfied by the CTS's in some subcategory of **CTS** with sufficient completeness properties. The axioms we have been able to discover are either too weak in the sense that the CTS's satisfying them are not very concrete, or else they are too strong in the sense that the resulting subcategory does not admit countable products.

As an example of what we are able to do, we obtain properties that characterize the CTS's obtained from trace automata. The result is patterned after Winskel's characterization of the domains of configurations of event structures defined by an enabling relation and a binary conflict relation [6, 18].

Define a CTS C to be *atomic* if the following holds:

- $t \uparrow v = u \uparrow v$ implies either $t = u$, $t = v$, or $u = v$, whenever t, u, and v are transitions of C, with t, v consistent and u, v consistent.

This is a very strong concreteness axiom which ensures that no nontrivial $\preceq$-relationships hold between transitions.

Lemma 3.12 *Suppose C is an atomic CTS. If t and u are coinitial transitions of C such that $t \preceq u$, then either $t = u$ or else $t = \mathrm{id}$.*

Proof – If $t \preceq u$, then $t \uparrow u = \mathrm{id}$, so $t \uparrow u = \mathrm{id} \uparrow u$. By the atomicity property, either $t = \mathrm{id}$, $u = \mathrm{id}$, or $t = u$. But $u = \mathrm{id}$ implies $t = u = \mathrm{id}$, so either $t = \mathrm{id}$ or $t = u$. ▮

If C is an atomic CTS, then we may define $\equiv$ to be the least equivalence relation on transitions of C such that $t \equiv t \uparrow u$ holds whenever t is an arbitrary transition and transition $u \neq t$ is a nonidentity transition consistent with t.

Theorem 10 *A CTS C is isomorphic to the CTS C_A associated with a trace automaton A iff the following conditions hold:*

1. *C is atomic.*
2. *For all transitions t, t' of C, if $t \equiv t'$ and $\mathrm{dom}(t) = \mathrm{dom}(t')$, then $t = t'$.*
3. *For all transitions $t \equiv t'$ and $u \equiv u'$, if the transitions t, u are consistent, and the transitions t', u' are coinitial, then t', u' are consistent as well.*

Proof – If C_A is the CTS associated with a trace automaton A, then it is a straightforward application of the definition of the residual operation on A to see that C_A has properties (1)-(3).

Conversely, suppose C is a CTS with the stated properties. Define the *events* of C to be the $\equiv$-equivalence classes $[t]$ of nonidentity transitions t of C. Let E be the set of all events of C, with concurrency relation $\|_E$ defined as follows: $[t]\|_E[u]$ iff there exist $t' \equiv t$ and $u' \equiv u$ such that t', u' are consistent and $t' \neq u'$. Clearly, the relation $\|_E$ is symmetric, and hypothesis (2) implies that it is irreflexive.

Define $A = (E, Q, T)$, where T contains all transitions $q \xrightarrow{\epsilon} q$ and all transitions $q \xrightarrow{[t]} r$ such that $t : q \to r$ is a nonidentity transition of C. One may now verify that A is an automaton, with $A \simeq C_A$. ∎

The equivalence, discussed in Section 3.5, from the category **Alph** of concurrent alphabets and weak morphisms to a full subcategory of **CTS**, suggests an interesting way to make the class of trace automata into a category **TrAuto**.

Formally, suppose $A = (E, Q, T)$ and $A' = (E', Q', T')$ are trace automata. Define a *morphism* from A to A' to be a pair $(\rho_{\mathrm{e}}, \rho_{\mathrm{s}})$, where $\rho_{\mathrm{e}} : E \to E'$ is a weak morphism of concurrent alphabets, and $\rho_{\mathrm{s}} : Q \to Q'$ is a function, such that the following holds:

- Suppose $q \xrightarrow{e} r \in T$, with $e \neq \epsilon$. Then for every enumeration $\{e'_1, \ldots, e'_n\}$ of $\rho_{\mathrm{e}}(\{e\})$, there exists a (necessarily unique) finite computation sequence

$$\rho_{\mathrm{s}}(q) = r'_0 \xrightarrow{e'_1} r'_1 \xrightarrow{e'_2} \ldots \xrightarrow{e'_n} r'_n = \rho_{\mathrm{s}}(r)$$

  of A'.

Let **TrAuto** denote the category of trace automata and their morphisms.

The map that takes a trace automaton A to the CTS $\widehat{C}_A$ extends to a functor $\mathrm{AuCts} : \mathbf{TrAuto} \to \mathbf{CTS}$. This functor does not yield an equivalence with a full subcategory of **CTS**, because $\mathrm{AuCts}(A)$ does not contain sufficient information to recover the concurrent alphabet of A up to isomorphism. Nevertheless, the category **TrAuto** seems quite interesting, and further study of it and its relationship to **CTS** seems worthwhile.

4 Conclusion

We have seen that by using the notion of a concurrent alphabet to introduce concurrency information into ordinary nondeterministic transition systems, we obtain a model of concurrent computation having a great deal of algebraic structure. The essential features of this structure can be expressed nicely with the help of the notion of the residual of one transition after another. The resulting algebra of residuals can be used to obtain useful insights into the capabilities and limitations of concurrency. In particular, the characterization of the domain of concurrent computations given by Theorem 4 leads at once to interesting characterizations (see [17]) of the input/output relations that are computable by various classes of concurrent automata.

References

[1] I. J. Aalbersberg and G. Rozenberg. Theory of traces. *Theoretical Computer Science*, 60(1):1–82, 1988.

[2] M. Bednarczyk. *Categories of Asynchronous Systems.* PhD thesis, University of Sussex, October 1987.

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

[4] G. Boudol. Computational semantics of term rewriting systems. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 169–236, Cambridge University Press. 1985.

[5] G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, XI:433–452, 1988.

[6] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming. Research Notes in Theoretical Computer Science*, Pitman, London, 1986.

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

[8] M. Kwiatkowska. *Categories of Asynchronous Systems.* PhD thesis, University of Leicester, May 1989.

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

[10] N. A. Lynch and E. W. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, July 1989.

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

[12] P. Panangaden and E. W. Stark. Computations, residuals, and the power of indeterminacy. In *Automata, Languages, and Programming*, pages 439–454, Springer-Verlag. Volume 317 of *Lecture Notes in Computer Science*, 1988.

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

[14] E. W. Stark. Compositional relational semantics for indeterminate dataflow networks. In *Category Theory and Computer Science*, pages 52–74, Springer-Verlag. Volume 389 of *Lecture Notes in Computer Science*, Manchester, U. K., 1989.

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

[16] E. W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989.

[17] E. W. Stark. On the relations computed by a class of concurrent automata. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, January 1990.

[18] G. Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.