Trade-offs in True Concurrency: Pomsets and Mazurkiewicz Traces

Bard Bloom*
Marta Kwiatkowska**

TR 91-1223
August 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported by NSF grant (CCR-9003441). bard@cs.cornell.edu.
**Supported by SERC GR/F 93050. This paper was partly written when the second author was Visiting Profeessor at CWI, Amesterdam, sponsored by the Netherlands Organisation for Scientific Research (NWO). mzk@uk.ac.le

Trade-offs in True Concurrency: Pomsets and Mazurkiewicz Traces

Bard Bloom*
Cornell University

Marta Kwiatkowska†
University of Leicester

August 5, 1991

Abstract

We compare finite pomsets and Mazurkiewicz traces, two models of true concurrency which generalize strings. We show that Mazurkiewicz traces are equivalent to a restricted class of pomsets. The restrictions lead to extra structure, with results analogous to the differences between simply typed and untyped languages. For example, traces are consistently complete in the prefix order, while pomsets are not; also, traces can be distinguished by observing sequences of actions, in contrast to the elaborate scheme required for distinguishing pomsets. Finally, we discuss the operations of sequential and parallel composition in the two models. This is part of an ongoing effort to relate models of concurrency.

1 Introduction

In the last two decades, a variety of models of concurrency have been proposed, covering a wide spectrum of powers of description, abstraction and precision. There are probably models well-suited for most problems that people encounter in practice. However, this theoretical wealth is in some disarray. The relations between models are not well understood. Ideally, there should be some grand catalog of models, complete with domains of appropriate use and relations to other models. This paper starts one piece

*Supported by NSF grant (CCR-9003441). bard@cs.cornell.edu

†Supported by SERC GR/F 93050. This paper was partly written when the second author was Visiting Professor at CWI, Amsterdam, sponsored by the Netherlands Organisation for Scientific Research (NWO). mzk@uk.ac.le

of this catalog, by describing the relation between Pratt's pomset model (20) and Mazurkiewicz's trace model (14). In brief, the relation between pomsets and Mazurkiewicz traces is analogous to that between *untyped* and *simply-typed* languages. Pomsets are more general and more powerful descriptive tools; Mazurkiewicz traces have more algebraic and conceptual structure.

Both pomsets and traces are based on the fundamental assumption that it must be possible to distinguish *concurrent* occurrences of actions from actions simply occurring in *either order*. The *interleaving* approach to semantics for concurrency, *e.g.*, CCS (17), does not consider this distinction essential. As a consequence, the interleaving approaches give rise to more abstract, elegant theories, whereas the true concurrency (also called *non-interleaving*) approaches are more expressive, but generally less abstract and more cumbersome to use. This paper is part of an ongoing search for an adequate denotational framework for true concurrency semantics, and is motivated by the evidence that the non-interleaving approach helps deal with issues such as confusion and fairness (10,11).

Pomsets are quite general and powerful descriptive tools; traces resemble pomsets with some restrictions on the form of concurrency. In Section 3, we make this similarity precise by giving a translation from traces to pomsets. We also describe the class of pomsets which are translations of traces in order-theoretic terms, as those pomsets which are both *irreflexive* and admit an *independency*. We also show that this translation is order-preserving.

Many models of concurrency (3,9,16,6,2,18) are described in terms of *observing processes*. In such models, two processes are different precisely if they can be distinguished in some experimental scenario. The scenarios range from simply observing the process run in isolation to communicating with it to duplicating it and beyond. Intuitively, the more complex and peculiar the experiments necessary to describe a model of concurrency, the less appropriate the model for simpler situations. Pratt and Plotkin (18) give a rather tricky experimental scenario for distinguishing pomsets, involving replacing individual actions by nondeterministic choices, and then having a large number of different observers watch the resulting pomset. In Section 4 we give the corresponding scenario for distinguishing Mazurkiewicz traces, which only requires watching the process run. This emphasizes how the restrictions on traces make them conceptually simpler and more tractable than pomsets.

In Section 5, we investigate the order-theoretic properties of traces and pomsets. There are several choices for partial orders on pomsets; one of

the simplest and most useful, as well as the most suitable for comparison to other models, is the prefix order. A prefix of a pomset or trace σ is another pomset or trace τ which can be extended to give σ; this corresponds to a possible partial execution of the events described by σ. For example, a consistent cut of a system execution (4,24) is simply a prefix of the system execution described as a pomset or trace.

The set of pomsets is not *consistently complete* in the prefix order; that is, there are two pomsets p_1 and p_2 which have a common upper bound, but no least upper bound. By contrast, Mazurkiewicz traces are consistently complete, and thus can be completed to a Scott domain (10,13). Because of these factors, traces should form an adequate framework for the denotational semantics of true concurrency. Denotational semantics is usually based on cartesian-closed categories of domains, of which Scott domains are an example. It remains to be seen if pomsets form such a category.

Finally, in Section 6, we discuss the algebras of pomsets and traces. Pomsets are closed under the quite natural operations of *parallel* $(p \,||\, q)$ and *sequential* $(p \,;\, q)$ composition: any two pomsets may be sequenced or run in parallel. This is algebraically pleasant, as well as useful in modeling and programming.

In the Mazurkiewicz trace view of the world, neither parallel nor sequential composition make sense in general. For example, if a and b are the action of a daisywheel printer printing the letters `a` and `b`, the trace $a \,||\, b$ does not make sense; the printer is incapable of printing two letters at the same time. However, $a \,;\, b$ makes perfectly good sense. Similarly, if c and d are actions on processors at opposite ends of the galaxy, the trace $c \,;\, d$ is not realizable; it is impossible to ensure that first c, then d happens without some communication between the two processors. However, $c \,||\, d$ is sensible. Accordingly, Mazurkiewicz traces are not closed under either parallel or sequential composition. In particular, neither is a total operation on traces. We give a simple typing scheme which describes when these operations can be used, and show that trace concatenation in fact models both. Our translation of Mazurkiewicz traces into pomsets allows us to define phrases like "$\cdot$ is used to express concurrency" formally.

In summary, pomsets and Mazurkiewicz traces are based on similar intuitions, that it is important to distinguish between concurrent actions and actions which may simply occur in either order. The pomset model is more general, allowing arbitrary orderings and concurrencies between events. The Mazurkiewicz model restricts the concurrencies possible, thereby gaining mathematical structure and philosophical simplicity at the expense of ex-

pressive power.

2 Definitions

In this section we briefly sketch the basic mathematics of pomsets and Mazurkiewicz traces. More complete presentations may be found in (19,20) and (14,15).

2.1 Preliminaries

Let Σ denote an alphabet of symbols (at most countable). $\Sigma^\star$ will denote the set of all finite strings ordered by the prefix order $\leq$. If s is a string, let $s[i]$ be the i'th symbol (*e.g.*, $abc[2] = b$); $s[i..j]$ is the substring from symbol i to symbol j inclusive; and $\#_a(s)$ be the number of preceding occurrences of the symbol a in s.

It is often useful to make the symbols in a string unique. If Σ is an alphabet, let Σ^u be the alphabet $\Sigma \times \mathbf{N}$. If $s \in \Sigma^*$, define *uniquification* s^u to be s with each symbol labelled by the number of occurrences of that symbol:

$$s^u[i] = \left\langle s[i], \left(\#_{s[i]}(s[1..i])\right)\right\rangle \tag{1}$$

For example, $abacb^u = \langle a,1\rangle\langle b,1\rangle\langle a,2\rangle\langle c,1\rangle\langle b,2\rangle$. Conversely, if t is a string in Σ^u, define its *agglomeration* t^g by stripping off all the integer tags in t. We extend both $\cdot^u$ and $\cdot^g$ pointwise to sets of strings, and so forth.

Lemma 2.1 *Let s, t be strings over Σ. Then we have the following:*

1. $s \leq t \iff s^u \leq t^u$
2. $s^{ug} = s$

Proof: Easy. ⏚

Unless otherwise stated, Σ will remain fixed throughout this paper.

2.2 Pomsets

A *pomset* is a generalized string in which (occurrences of) symbols may be unordered in an arbitrary way.

Definition 2.2 *A* Σ-labelled partial order *is a tuple $\langle E, \leq, \mu\rangle$, where E is a (finite) set, $\leq$ is a partial order on E, and $\mu : E \to \Sigma$.*

Elements of E are called *events* and considered to be things happening; elements of Σ are *actions*, and are the things that happen. $e_1 \leq e_2$ is intended to mean that e_1 occurs before e_2 in time. If e_1 and e_2 are incomparable (notation $e_1 \perp e_2$), then they occur in indeterminate order or in parallel. $\mu(e)$ is the action that happened at event e. For example, $e_1 \leq e_2$ could represent two successive presses of the a key on the keyboard, in which case we would have $\mu(e_1) = \mu(e_2) = \mathtt{a}$. However, given two keyboards, it would be feasible for two presses of a keys to happen in parallel; we would then have $e_1 \perp e_2$ with $\mu(e_1) = \mu(e_2) = \mathtt{a}$.

Two labelled partial orders are *equivalent* if they differ only in the names of events; that is, they must have the same alphabet of actions, and order- and label-isomorphic sets of events. We identify equivalent labelled partial orders:

Definition 2.3 *A* pomset $[E, \leq, \mu]$ *is the isomorphism class of the* Σ*-labelled partial order* $\langle E, \leq, \mu \rangle$.

We shall often take the liberty of confusing a pomset p with a convenient representative $\langle E_p, \leq_p, \mu_p \rangle$.

Strings are equivalent to pomsets in which $\leq$ is a total order; multisets are equivalent to pomsets in which $\leq$ is equality.

There are a number of standard operations on pomsets, of which we will be concerned with $p \parallel q$, which exhibits behavior p and q concurrently, and $p\,;q$, which acts first like p and then like q.

Definition 2.4 *Let* $p = [E, \leq, \mu]$ *and* $p' = [E', \leq', \mu']$ *such that* $E \cap E' = \emptyset$. *Define the pomsets* $p \parallel p'$ *and* $p\,;p'$ *by:*

$$\begin{aligned} p \parallel p' &= [E \cup E', (\leq_p \cup \leq_{p'}), \mu \cup \mu'] \\ p\,;p' &= \left[E \cup E', (\leq_p \cup \leq_{p'} \cup (E \times E')), \mu \cup \mu'\right] \end{aligned}$$

Examples may be found in Figure 1. Note that $\parallel$ is only the most basic form of concurrency, as there is no communication between the components. Concurrency involving communication will be discussed elsewhere.

We now define prefix ordering on pomsets. If r and s are strings, r is a prefix of s if we may add symbols after those in r to obtain s. By analogy, p is a prefix of q if we may add events after *and independent to* those of p to obtain q.

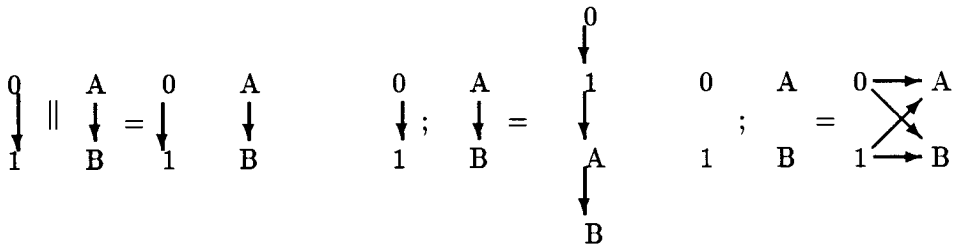

Figure 1: Sequencing and Concurrency of Simple Pomsets

Definition 2.5 *The pomset p is a* prefix *of q, written $p \leq q$, iff there is an order-and-label-preserving injection of p into q with a downward-closed range; that is, if $e_1 \leq e_2$ are events of q, and e_2 is in the range of the injection, then so is e_1.*

It is easy to see that $\leq$ is a partial order.

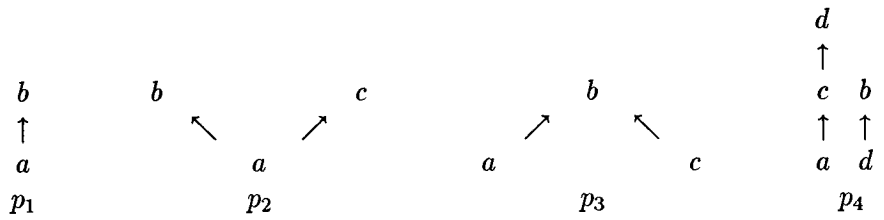

Figure 2: Examples of Prefix Order for Pomsets: $p_1 \leq p_2, p_1 \not\leq p_3, p_1 \not\leq p_4$.

Definition 2.6 *A pomset q is a* linearization *of a pomset p if there exist representatives $\langle E, \leq, \mu \rangle$ and $\langle E, \leq', \mu \rangle$ of p and q respectively such that $\leq \subseteq \leq'$ and $\leq'$ is a total order. The set of all linearizations of p will be denoted* $\mathsf{lin}(p)$.

Given a linearization $q = [E, \leq, \mu]$ of the pomset p, we can obtain a string of symbols in Σ inductively by concatenating the symbols in $\mu(E)$ in a manner consistent with the order $\leq$. If it is clear from the context, a string thus obtained from a linearization q of a pomset p will also be called a *linearization* of p. This identification can be easily justified because each string linearization determines a pomset linearization; just take as the set of events the symbols in s^u with the obvious order.

2.3 Mazurkiewicz Traces

Mazurkiewicz traces (called simply "traces" from now on) are also generalizations of strings to allow the representation of non-sequential behavior. Traces are equivalence classes of strings with respect to a congruence relation that allows to commute certain pairs of consecutive symbols.

Definition 2.7 *Let Σ be a countable alphabet. An* independency *is a symmetric, irreflexive relation $\iota \subseteq \Sigma \times \Sigma$. Call an ordered pair $\langle \Sigma, \iota \rangle$ a* concurrent alphabet *if Σ is an alphabet and $\iota \subseteq \Sigma \times \Sigma$ is an independency relation.*

So, $a \ \iota \ b$ holds iff a and b cannot be causally related, in which case they may happen concurrently. For example, if $a \ \iota \ b$, then the strings *cabd* and *cbad* are viewed as representing two sequential observations of the same non-sequential behavior. Note that no action can be concurrent with itself (independency is irreflexive) and concurrency is always mutual (independency is symmetric). In typical use, actions on different ports or processors are independent, while those in the same place are dependent. Actions on a bus connecting two places will be dependent on actions local to each place; this is why ι is not an equivalence relation.

More formally,

Definition 2.8 *Let $\langle \Sigma, \iota \rangle$ be a concurrent alphabet and s, t be finite strings of symbols from Σ. We say s and t are* Mazurkiewicz equivalent, *notation $s \equiv_\iota t$, iff s can be transformed to t by a finite number of exchanges of adjacent, independent actions. A* Mazurkiewicz trace $\sigma = [s]_\iota$ *over the concurrent alphabet $\langle \Sigma, \iota \rangle$ is the Mazurkiewicz equivalence class of a string.*

Let $\Sigma = \{a, b, c\}$ with $a \ \iota \ b$ and $b \ \iota \ c$ (and symmetrically) as the only independencies. Then we have:

$$\begin{aligned} [ab]_\iota &= \{ab, ba\} \\ [aba]_\iota &= \{aba, baa, aab\} \\ [abc]_\iota &= \{abc, bac, acb\} \\ [abca]_\iota &= \{abca, baca, acba, acab\} \end{aligned}$$

The theory of traces has been developed in (14,21,1,10). We present only a few facts:

1. Concurrent alphabets amount to types. Traces over the same concurrent alphabet are compatible; traces over different concurrent alphabets are incompatible, and we should make no attempt to combine them.

2. Catenation of traces with the same independency is well-defined by $[s]_\iota \cdot [t]_\iota = [st]_\iota$, and indeed traces over a given concurrent alphabet form a cancellative monoid with $[\epsilon]_\iota$ as the unit.

3. We may define prefix ordering as usual: $\sigma \sqsubseteq \tau$ iff $\exists \gamma . \tau = \sigma \cdot \gamma$. Prefixing is a partial order, but not a total order: $[a]_\iota$ and $[b]_\iota$ are both prefixes of $[ab]_\iota$.

4. $\sigma \sqsubseteq \tau$ iff $\forall x \in \sigma \exists y \in \tau . x \leq y$.

3 Translations between Traces and Pomsets

Both pomsets and traces may be informally described as "generalizations of strings which allow for some actions to happen concurrently." Traces are somewhat more constrained than pomsets: if a and b occur in parallel in a trace, they cannot occur sequentially; pomsets have no such restriction. In this section, we give two equivalent translations from traces to pomsets (Theorems 3.10 and 3.17). There are pomsets which do not correspond to any trace; we give necessary and sufficient conditions under which pomsets correspond to traces (Theorem 3.19).

3.1 Traces as Pomsets

The partial order of prefixes of $[abab]_\iota$ (where $a \; \iota \; b$) is presented in Figure 3. We shall extract a pomset (in this case, $aa \parallel bb$) in two equivalent ways (see Figure 4). Notice that the set of linearizations of $aa \parallel bb$ is equal to the set $[abab]_\iota$. The theorems in this section generalize this observation.

We present two ways of converting a trace to a pomset, and show that they are equivalent. The first is fairly direct, turning the symbols of σ into events. It essentially uses a theorem of Szpilrajn (23), that a partial order is equal to the intersection of its linearizations. A similar construction was also used in (15).

If $\langle \Sigma, \iota \rangle$ is a concurrent alphabet, let $\langle \Sigma^u, \iota^u \rangle$ be the concurrent alphabet with $\iota^u = \left\{ \langle \langle a, i \rangle, \langle b, j \rangle \rangle | a \; \iota \; b \right\}$.

Lemma 3.1 *Let s be a string over Σ. Then $[s^u]_{\iota u} = ([s]_\iota)^u$.*

Proof: Easy. ⏚

Definition 3.2 *Given a trace $\sigma = [s]_\iota$, we define* $\mathsf{pom}^\star(\sigma)$ *to be the pomset with*

- *$E_{\sigma\star}$ the set of symbols $\langle a, i\rangle$ in s^u.*
- *$\langle a, k\rangle \leq_{\sigma\star} \langle b, l\rangle$ if, for every $t \in \sigma$, $\langle a, k\rangle$ precedes or equals $\langle b, l\rangle$ in t^u.*
- *$\mu_{\sigma\star}(\langle a, k\rangle) = a$.*

For example, the trace $[abab]_\iota$ in Figure 3 gives $E_{[abab]} = \{\langle a, 1\rangle, \langle a, 2\rangle, \langle b, 1\rangle, \langle b, 2\rangle\}$ with $\langle a, 1\rangle \leq \langle a, 2\rangle$ and $\langle b, 1\rangle \leq \langle b, 2\rangle$ as the only strict orderings.

The second conversion requires somewhat more machinery, but has been more useful in proofs.

Definition 3.3 *A* point *of a trace $[s]_\iota$ is a non-empty trace prefix $[r]_\iota$ of $[s]_\iota$ such that all representatives $r' \in [r]_\iota$ end in the same symbol. The* action *at a point $[r]_\iota$,* $\mathsf{act}([r]_\iota)$*, is the common final symbol of all the representatives.*

The points of $[abab]_\iota$ are $[a]_\iota$, $[aa]_\iota$, $[b]_\iota$, and $[bb]_\iota$, with actions a, a, b, b respectively. $[aab]_\iota$ is not a point, as it contains both aab and aba. The points[1] of a trace are suitable events for the corresponding pomset:

Definition 3.4 *Given a trace $\sigma = [s]_\iota$, we define* $\mathsf{pom}(\sigma)$ *to be the pomset with*

- *E_σ the set of points of σ;*
- *$\leq_\sigma$ the restriction of the trace prefix order to E_σ;*
- *μ_σ the map* act.

We now prove that the two constructions of pomsets from traces coincide. We shall require the following lemmas.

Lemma 3.5 *The points of $[s^u]_{\iota u}$ are precisely the uniquifications of the points of $[s]_\iota$, and $\cdot^u$ is an order-isomorphism between them.*

[1] Points of a trace are also *primes* (21).

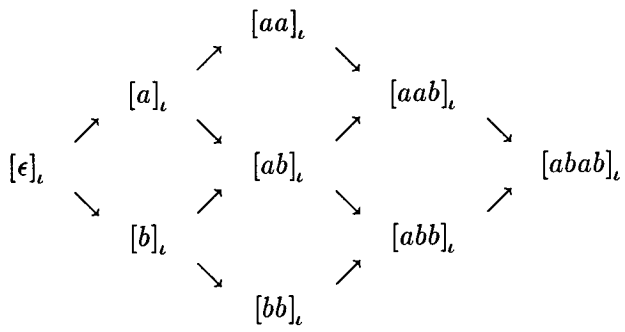

Figure 3: Partial order of prefixes of $[abab]_\iota$

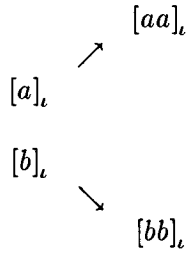

Figure 4: Sub-partial order of points of $[abab]_\iota$

Proof: It is easy to see that $[pa]_\iota$ is a point of $[s]_\iota$ iff $[pa^u]_{\iota^u}$ is a point of $[s^u]_{\iota^u}$.

For order-isomorphism, we reason thus:

$$\begin{aligned} [pa]_\iota \sqsubseteq [qb]_\iota &\iff \\ \forall x \in [pa]_\iota, \exists y \in [qb]_\iota . x \leq y &\iff \\ \forall x^u \in {[pa]_\iota}^u, \exists y \in {[qb]_\iota}^u . x^u \leq y^u &\iff \\ {[pa]_\iota}^u \sqsubseteq {[qb]_\iota}^u & \end{aligned}$$

Lemma 3.6 *Let t be a uniquely-labelled string (that is, one in which no symbol appears twice), and suppose that $\neg(a \iota b)$, and a precedes b in t. Then a precedes b in every representative t' of $[t]_\iota$.*

Proof: Suppose there exists $t' \in [t]_\iota$ such that b precedes a in t'. Then there must exist a finite sequence of exchanges of consecutive independent symbols of t' which gives t. Thus, the order of a and b would have to be changed, but $\neg(a \iota b)$ and we have reached a contradiction. $\Lightarrow\!\Leftarrow$

Lemma 3.7 *Let $[ra]_\iota$ and $[sb]_\iota$ be points of some trace $[t]_\iota$, such that $\neg(a \iota b)$. Then $[ra]_\iota$ and $[sb]_\iota$ are comparable.*

Proof: Suppose $[ra]_\iota{}^u$ and $[sb]_\iota{}^u$ are distinct points of $[t^u]_{\iota^u}$ (the result follows trivially if they are equal). Let $n_a = \#_a(ra)$ and $n_b = \#_b(sb)$. Suppose without loss of generality that $\langle a, n_a\rangle$ precedes $\langle b, n_b\rangle$ in t^u. Then by Lemma 3.6, $\langle a, n_a\rangle$ precedes $\langle b, n_b\rangle$ in every element of $[t^u]_{\iota^u}$. Thus $[ra]_\iota{}^u \sqsubseteq [sb]_\iota{}^u$. By the order-isomorphism (Lemma 3.5), we know that $[ra]_\iota \sqsubseteq [sb]_\iota$ as desired. $\Lightarrow\!\Leftarrow$

Lemma 3.8 *Let $\tau = [t]_\iota$ be a trace. Suppose that $n \leq \#_a(t)$. Then*

1. *There is a unique point $\mathsf{point}(\tau, a, n) = \rho \sqsubseteq \tau$ such that $\#_a(\rho) = n$ and $\mathsf{act}(\rho) = a$.*
2. *If $\sigma \sqsubseteq \tau$ and $\#_a(\sigma) \geq n$ then $\mathsf{point}(\tau, a, n) \sqsubseteq \sigma$.*

Proof: For existence, let r be any minimum-length element of

$$R = \{r \mid \exists t' \in \tau.(r \leq t' \wedge \#_a(r) = n)\}$$

R is nonempty because $\#_a(t) \geq n$. Let $\rho = [r]_\iota$. Clearly any $r' \in \rho$ ends in a; if it did not, we could find a shorter element of R by removing the non-a tail. Thus ρ is a point as desired.

For uniqueness, suppose that ρ' were another such point. By Lemma 3.7, we know that ρ and ρ' are comparable. As r is minimum-length, we must have $\rho \sqsubseteq \rho'$, and hence $\rho\sigma = \rho'$ for some σ. σ cannot contain any a's, as $\#_a(\rho) = n = \#_a(\rho')$. However, σ cannot contain any symbols other than a either, or else ρ' would not be a point ending in a. Hence σ is empty and so $\rho = \rho'$.

For the second part, suppose that $\#_a(\sigma) \geq n$. Let $\rho' = \mathsf{point}(\sigma, a, n)$. Then $\rho' \sqsubseteq \sigma \sqsubseteq \tau$. By uniqueness of $\mathsf{point}(\tau, a, n)$, we have $\rho = \rho'$. $\Lightarrow\!\Leftarrow$

Corollary 3.9 *If π is a point of τ with $\mathsf{act}(\pi) = a$, then $\pi = \mathsf{point}(\tau, a, \#_a(\pi))$.*

Proof: From uniqueness in Lemma 3.8 part 1. ⏚

Finally, we can show the main result of this section.

Theorem 3.10 *For any trace* $\tau = [t]_\iota$, $\mathsf{pom}(\tau) = \mathsf{pom}^\star(\tau)$.

Proof: To prove that two pomsets are equal, we must exhibit an order– and label–preserving isomorphism between them. Define $f : E_\tau \to E_{\tau\star}$ by:

$$f([pa]) = \langle a, \#_a(pa)\rangle$$

This is well-defined and 1-1 because, by Lemma 3.8, if $[pa]_\iota$ and $[qa]_\iota$ are points with the same number of occurrences of a, then $[pa]_\iota = [qa]_\iota$.

1. (Onto) If $\#_a(t) \geq n$, then $\mathsf{point}(\tau, a, n)$ exists. By definition, $f(\mathsf{point}(\tau, a, n)) = \langle a, n\rangle$.

2. (Order-preserving $\Rightarrow$) We must prove $[pa]_\iota \sqsubseteq [qb]_\iota \Rightarrow f([pa]_\iota) \leq_{\tau\star} f([qb]_\iota)$. Suppose $[pa]_\iota \sqsubseteq [qb]_\iota$, $f([pa]_\iota) = \langle a, n_a\rangle$, and $f([pb]_\iota) = \langle b, n_b\rangle$. Suppose also that $\langle a, n_a\rangle \not\leq_{\tau\star} \langle b, n_b\rangle$; then there must be some element $x\langle b, n_b\rangle y\langle a, n_a\rangle z$ of $[t]_\iota{}^u$.

   As $\langle b, \_\rangle$'s do not commute, we have $\#_b(x^g b) = n_b$, and so by Lemma 3.8 we have $[qb]_\iota \sqsubseteq [x^g b]_\iota$. So, $\#_a(qb) \leq \#_a(x^g b) < n_a$. But $n_a = \#_a(pa) \leq \#_a(qb) < n_a$, which is impossible. Thus $f([pa]_\iota) \leq_{\tau\star} f([qb]_\iota)$.

3. (Order-preserving $\Leftarrow$) We must prove $f([pa]_\iota) \leq_{\tau\star} f([qb]_\iota)$ implies $[pa]_\iota \sqsubseteq [qb]_\iota$. Suppose that $\langle a, n_a\rangle \leq_{\tau\star} \langle b, n_b\rangle$. Without loss of generality suppose $\langle a, n_a\rangle \neq \langle b, n_b\rangle$ (the implication follows trivially if they are equal). Then every prefix of $[t]_\iota$ with at least n_b occurrences of b also has at least n_a occurrences of a.

   In particular, $[qb]_\iota$ has exactly n_b b's, hence it has at least n_a a's. By Lemma 3.8 part 2, we know that $\mathsf{point}(\tau, a, n_a) \sqsubseteq [qb]_\iota$. By Corollary 3.9, we know that $[pa]_\iota = \mathsf{point}(\tau, a, n_a)$. So $[pa]_\iota \sqsubseteq [qb]_\iota$ as desired.

4. (Label-preserving) Clear.

⏚

Figure 5: Irreflexive and Independence Examples

3.2 Pomsets as Traces

A trace is, first of all, a set of strings. We may attempt to convert a pomset p into a trace by finding the set of strings which correspond to it, *viz.* its set of linearizations $\mathsf{lin}(p)$. If this is to be a trace, it must respect some independency ι. The obvious choice is ι_p:

Definition 3.11 *If p is a pomset, then the* natural independency of p, ι_p, *is:*

$$a \; \iota_p \; b \qquad \textit{iff} \qquad \exists e_a, e_b \in E_p.(\mu(e_a) = a \wedge \mu(e_b) = b \wedge e_a \perp e_b) \quad (2)$$

In this section, we give conditions under which this is an independency and p truly respects it; in particular, under which $\mathsf{lin}(\mathsf{pom}(\sigma)) = \sigma$.

It is not always possible to translate a pomset into a trace over the same alphabet. There are two canonical counterexamples, p_{irr} and p_{indep}, given in Figure 5. p_{irr} cannot be a trace, as the first a causes b and the second is independent of it: the independence relation in a trace must be obeyed consistently over the entire trace. The relation $\iota_{p_{irr}}$ is an independency, but $\mathsf{lin}(p_{irr})$ does not obey it. p_{indep} cannot be a trace, as two a's occur concurrently in p_{indep} and all occurrences of the same action in a trace must be causally related; $\iota_{p_{indep}}$ is not irreflexive and hence not even an independency. We now show that these are the only ways that pomsets can fail to be equivalent to traces.

Definition 3.12 *A pomset is* irreflexive[2] *iff, whenever e_1 and e_2 are distinct events with $\mu(e_1) = \mu(e_2) = a$, then either $e_1 \leq e_2$ or $e_1 \geq e_2$.*

[2] A related notion is that of a *semiword* (8,22).

An irreflexive pomset corresponds to a trace if the independence relation ι_p really behaves like an independence; that is, if whenever $a\ \iota_p\ b$, then a should not cause b. We detect causality by *immediate precedence*. Formally,

Definition 3.13 *If $\leq$ is a partial order, then a* immediately precedes *b, written $a \leq^! b$, iff $a < b$ and there is no c with $a < c < b$.*

Definition 3.14 *A pomset p is* independent *iff there are no events e_a, e'_a, e_b, and e'_b such that*

$$\begin{array}{llll} \mu(e_a) = & \mu(e'_a) & = a & \qquad e_a \leq^! e_b \\ \mu(e_b) = & \mu(e'_b) & = b & \qquad e'_a \perp e'_b \end{array}$$

For example, p_{irr} is irreflexive but not independent; p_{indep} is independent but not irreflexive.

A pomset does *not* uniquely determine an independency relation. This is why we shall need the following notion.

Definition 3.15 $\mathrm{POM}(\iota)$, *the set of all pomsets compatible with the independency ι, is the set of irreflexive pomsets such that:*

- *Whenever $e \perp e'$, then $\mu(e)\ \iota\ \mu(e')$, and*
- *Whenever $e \leq^! e'$, then $\neg(\mu(e)\ \iota\ \mu(e'))$.*

It is clear that all elements of $\mathrm{POM}(\iota)$ are irreflexive, independent pomsets.

For example, $a \parallel b \in \mathrm{POM}(\iota)$ if $\iota = \{(a,b),(b,a),(b,c),(c,b)\}$. Since c does not occur in $a \parallel b$, there is no evidence of the fact that $b\ \iota\ c$. Note that $a \parallel b \in \mathrm{POM}(\iota')$, where the independency $\iota' = \{(a,b),(b,a)\}$.

Theorem 3.17 precisely characterizes the translation from traces to pomsets. We shall require the following lemma.

Lemma 3.16 *If $a\ \iota\ b$ and $[ra]_\iota \sqsubset [sb]_\iota$ are points, then there is a point $[pc]_\iota$ with $[ra]_\iota \sqsubset [pc]_\iota \sqsubset [sb]_\iota$.*

Proof: Note that $[ra]_\iota[xb]_\iota = [sb]_\iota$. There must be some element c of x (let c be the first if there are several, and let n_c be the number of this occurrence of c) such that $\neg(c\ \iota\ a)$. (If not, then $[sb]_\iota = [raxb]_\iota = [rxab]_\iota = [rxba]_\iota$ is not a point). Let $[pc]_\iota = \mathsf{point}([sb]_\iota, c, n_c)$ be the shortest prefix of $[sb]_\iota$ containing that c. The last a in $[ra]_\iota$ must also occur in $[p]_\iota$, because it cannot commute with c. Everything before it, *i.e.* r, must also occur in p because none of these symbols can commute with a. So, $[ra]_\iota \sqsubset [pc]_\iota$. As $[pc]_\iota \sqsubseteq [rax]_\iota \sqsubset [sb]_\iota$, we know that $[pc]_\iota$ is the desired point. $\clubsuit$

We can now prove one of the main theorems of this section.

Theorem 3.17 *For any string s and independency relation ι,*

1. $\mathsf{pom}([s]_\iota) \in \mathrm{POM}(\iota)$, *and*
2. $\mathsf{lin}(\mathsf{pom}([s]_\iota)) = [s]_\iota$

Proof:

1. We need to show that $\mathsf{pom}([s]_\iota)$ is an irreflexive pomset which is compatible with ι. Irreflexivity follows from Lemma 3.7; note that $\neg(a \ \iota \ a)$. For compatibility with ι, consider two points $[pa]_\iota$ and $[qb]_\iota$.
   - If $[pa]_\iota \perp [qb]_\iota$ then $a \ \iota \ b$ by Lemma 3.7.
   - If $[pa]_\iota \sqsubseteq^! [qb]_\iota$ then we must show $\neg(a \ \iota \ b)$. Suppose the converse: $a \ \iota \ b$. Since $[pa]_\iota \sqsubset [qb]_\iota$ and $a \ \iota \ b$, then by Lemma 3.16 there is a point $[rc]_\iota$ with $[pa]_\iota \sqsubset [rc]_\iota \sqsubset [qb]_\iota$. But this violates the assumption that $[pa]_\iota \sqsubseteq^! [qb]_\iota$.
2. We have shown in the first part of this proof that $\mathsf{pom}([s]_\iota) \in \mathrm{POM}(\iota)$. Let $P = \mathsf{pom}([s]_\iota)$. We must show that (a) $\mathsf{lin}(P)$ is closed under exchanges of adjacent, independent symbols, (b) all strings in $\mathsf{lin}(P)$ are Mazurkiewicz equivalent with respect to the independency ι, and (c) $\mathsf{lin}(\mathsf{pom}([s]_\iota)) = [s]_\iota$.
   (a) (Closure under exchanges of adjacent independent symbols) Let e_a, e_b be events with $\mu(e_a) = a, \mu(e_b) = b$ and $a \ \iota \ b$, and let $x \in \mathsf{lin}(P)$ such that $x = x'abx''$. Since P is in $\mathrm{POM}(\iota)$, then $\neg(e_a \leq^! e_b)$. Thus, either $e_a \perp e_b$, or $e_a < e_c < e_b$ for some e_c. The latter case is impossible because a and b are adjacent. Hence, $e_a \perp e_b$, so a and b can be interchanged: $x'bax'' \in \mathsf{lin}(P)$.
   (b) (Equivalence of strings in $\mathsf{lin}(P)$) It suffices to show that, if $xaby$ is in $\mathsf{lin}(P)$ and $\neg(a \ \iota \ b)$ and $a \neq b$, then $xbay$ is *not* in $\mathsf{lin}(P)$. Since the pomset P is irreflexive, the events labelled with a are totally ordered, so we can assign occurrence numbers k. Suppose $xaby$ comes from the linearization Q, where the a and b come from e_a and e_b. Since $\neg(a \ \iota \ b)$, then e_a and e_b are comparable in P, and, since e_a comes first in the linearization, then $e_a \leq_P e_b$. Suppose that $xbay$ came from some other linearization Q'. There are k occurrences of a in xab, and hence k occurrences of a in xba. So the event that determines a is e_a. There are l occurrences of b in xab, and hence in xba. So the event that determines b is e_b. Thus, $e_b \leq_{Q'} e_a$ and $e_a \leq_P e_b$. This is impossible.

(c) ($\mathsf{lin}(\mathsf{pom}([s]_\iota)) = [s]_\iota$) We have shown that $\mathsf{lin}(P) \in \mathrm{POM}(\iota)$, for $P = \mathsf{pom}[s]_\iota$, is a trace over ι. Clearly, s is a linearization of $\mathsf{pom}([s]_\iota)$, and the desired result follows.

$\clubsuit$

Corollary 3.18 *If σ is a trace, then $\mathsf{pom}(\sigma)$ is an irreflexive, independent pomset.*

Proof: Since $\mathsf{pom}(\sigma) \in \mathrm{POM}(\iota)$ by Theorem 3.17, then by definition it must be irreflexive and independent. $\clubsuit$

The following precisely characterizes the translation from irreflexive, independent pomsets to traces.

Theorem 3.19 *The following are equivalent:*

1. *p is an irreflexive, independent pomset;*
2. *$p \in \mathrm{POM}(\iota_p)$;*
3. *$\mathsf{lin}(p)$ is a trace with some independency, and $p = \mathsf{pom}(\mathsf{lin}(p))$;*
4. *$\mathsf{lin}(p)$ is a trace with independency ι_p, and $p = \mathsf{pom}(\mathsf{lin}(p))$.*

Proof:

1. ($1 \Rightarrow 2$) Suppose p is an irreflexive and independent pomset. If $a \; \iota_p \; b$, then by independence $\not\exists\left(e_a \leq^! e_b\right).\mu(e_a) = a \wedge \mu(e_b) = b$; so $p \in \mathrm{POM}(\iota_p)$.

2. ($2 \Rightarrow 3$) Suppose $p \in \mathrm{POM}(\iota_p)$. Then $\mathsf{lin}(p)$ is a trace with independency ι_p by Theorem 3.17. Also note that $\mathsf{lin}(p)$ is a trace with *any* independency $\iota \subseteq \Sigma \times \Sigma$ containing ι_p.
   We show that $p = \mathsf{pom}(\mathsf{lin}(p))$ by constructing an order- and label-preserving isomorphism between them. Choose $e \in E_p$. Let $a = \mu(e)$ and define:

$$f(e) = \mu(\mathsf{lin}(\downarrow e))$$

   where $\downarrow e = \{e' \in E_p \mid e' \leq_p e\}$. (That is, take the set of all linearizations of the set of events below e, and convert them to strings over Σ). The following are easy observations:

- $\downarrow e$ is an irreflexive, independent pomset;
- $\mathsf{lin}(\downarrow e)$ is a trace prefix of $\mathsf{lin}(p)$ (to see this, consider the linearization in which events in $\downarrow e$ appear first);
- Every linearization of $\downarrow e$ ends with a;
- Thus $f(e)$ is a point with action a.

It remains to show that f is an order-isomorphism.

- (1-1) Suppose that $f(e) = f(e')$ for distinct e, e'. Then $e < e'$ is impossible by length arguments (and, symmetrically, $e' < e$ is also impossible). Suppose $e \perp e'$. Then $\mu(e) = \mu(e')$, which violates irreflexivity, and we have reached a contradiction.
- (Onto) If there are n events in p, then there are also n points in $\mathsf{lin}(p)$, and each point is in the range of f.
- (Order-preserving $\Rightarrow$) If $e \leq e'$, then $\downarrow e \subseteq \downarrow e'$. Hence also $f(e) \leq f(e')$.
- (Order-preserving $\Leftarrow$) Suppose $f(e) \leq f(e')$, but $e \not\leq e'$. Let $a = \mu(e)$, $b = \mu(e')$, then either $e > e'$ or $e \perp e'$.
  - (Case $e > e'$) This would imply $f(e) > f(e')$ by the (1-1) and (Order-preserving $\Rightarrow$) parts of this proof, which contradicts the assumption $f(e) \leq f(e')$.
  - (Case $e \perp e'$) Then $a \iota_p b$. Let k be the number of occurrences of a in $f(e)$, l the number of occurrences of b in $f(e')$. Let:

$$\begin{aligned} D &= \downarrow e \cap \downarrow e' \\ F &= \downarrow e \setminus \downarrow e' \\ F' &= \downarrow e' \setminus \downarrow e \end{aligned}$$

    Then $D \leq F \cup F'$, and $F \perp F'$. Consider a linearization of p in which all events of D precede F, and all events of F precede F'. There are fewer than k occurrences of a in D (since there are k in $D \cup F$ and at least 1 in F). There are no occurences of a in F' (since $F' \perp e$, and the pomset p is irreflexive). Similarly, there are no occurrences of b in F. So, there is a prefix (a linearization of D followed by F') which contains l occurrences of b and fewer than k occurrences of a. This contradicts $f(e) \leq f(e')$.

3. (3 $\Rightarrow$ 4) Clear.

4. (4 $\Rightarrow$ 1) By Theorem 3.17.

$\clubsuit$

It is worth stressing that the set of linearizations of a given independent and irreflexive pomset does not uniquely determine an independency relation. For example, the pomset $a\,;b\,;c$ admits abc as the only linearization. The set $\{abc\}$ is a valid trace not only for empty independency, but also for one in which $a\,\iota\,c$.

The translations pom and lin are well-behaved on their domains. We show that they preserve order.

Lemma 3.20 *If* $[s]_\iota \sqsubseteq [t]_\iota$ *then* $\mathsf{pom}([s]_\iota)$ *is the restriction of* $\mathsf{pom}([t]_\iota)$ *to the set* $E_{[s]_\iota}$ *of events of* $\mathsf{pom}([s]_\iota)$.

Proof: Suppose $[s]_\iota \sqsubseteq [t]_\iota$, and let $[pa]_\iota \sqsubseteq [s]_\iota$. Then $[pa]_\iota \sqsubseteq [t]_\iota$ and all representatives of $[pa]_\iota$ end in a. Hence, $[pa]_\iota$ is a point of $[t]_\iota$, and thus $E_{[s]_\iota} \subseteq E_{[t]_\iota}$. The remainder is clear. $\clubsuit$

Theorem 3.21

1. *For all traces* σ, τ: $\sigma \sqsubseteq \tau \Rightarrow \mathsf{pom}(\sigma) \leq \mathsf{pom}(\tau)$.

2. *For all pomsets* $p, q \in \mathrm{POM}(\iota)$: $p \leq q \Rightarrow \mathsf{lin}(p) \sqsubseteq \mathsf{lin}(q)$.

Proof:

1. Let $[s]_\iota \sqsubseteq [t]_\iota$. To show $\mathsf{pom}([s]_\iota) \leq \mathsf{pom}([t]_\iota)$ we need to construct an order- and label-preserving injection with a downward-closed range. Let e be an event of $\mathsf{pom}([s]_\iota)$ and define:

$$\begin{aligned} f(e) &= e \\ \mu(f(e)) &= \mu(e) \end{aligned}$$

   It is easy to see by Lemma 3.20 that f is the desired injection.

2. Without loss of generality, suppose $p, q \in \mathrm{POM}(\iota)$ such that $E_p \subseteq E_q$ and $p \leq q$, in which case E_p is a downward-closed subset of E_q. Let $F_p = E_q \setminus E_p$. Then $d \in F_p$ implies either $d \perp E_p$ or $E_p \leq d$.

We now construct a linearization of q as follows. Choose any linearization l_1 of E_p (hence of p), and any linearization l_2 of F_p. The 'concatenation' $l_1 l_2$ is a linearization of $E_p F_p$, and hence also of q. It now follows that $\mu(l_1 l_2) \in \mathsf{lin}(q)$, $\mu(l_1) \in \mathsf{lin}(p)$, from which we have by Theorem 3.19:

$$\mathsf{lin}(p) = [\mu(l_1)]_\iota \sqsubseteq [\mu(l_1 l_2)]_\iota = \mathsf{lin}(q)$$

as desired.

We have identified the traces as those pomsets which are both irreflexive and independent. There are two obvious intermediate classes, the irreflexive pomsets and the independent ones. The structure of these classes largely remains to be investigated.

4 Observing Pomsets and Traces

Many notions of process equivalence (3,9,16,6,2) are most clearly understood as equivalence with respect to some set of *experiments*. In brief, two processes should be equivalent if and only if they cannot be told apart no matter how they are used. This definition has two parameters: methods for telling processes apart (the *experiments*) and methods for using them (*e.g.*, a programming language). Many, but not all, standard process equivalences can be understood as equivalence with respect to a reasonable set of experiments.

Typically, a process is considered a "black box" or an "alien calculator," an opaque shell surrounding some incomprehensible internal state, with only a few controls and indicators of unknown function and significance poking through. The experimenter performs some sort of experiment on the process, manipulating the controls and observing the indicators. Two processes are equivalent if they yield the same results on all experiments in all contexts. In many cases, it suffices to consider equivalence with respect to all experiments alone, as it frequently implies equivalence with respect to all experiments in all contexts as well.

For the moment, we consider the problem of distinguishing distinct pomsets and traces. These are used as the basic notions of "system execution" in the two theories. It would be rather distressing if there were indistinguishable yet different system executions possible. (For one thing, it would

be possible to specify that a program should do one but not the other of these things; such a specification would be difficult to justify or understand.) In fact, distinct system executions are distinguishable in both models. The scenarios build on one of the simplest schemes, in which processes are considered to be black boxes with lights (one per action) and a start button. When the start button is pushed, lights will flash for some time; the observer watches the flashing, and writes down what she sees. When events are unordered, the lights may flash concurrently or in any order; furthermore, they may be so far apart that different observers will disagree about what order they flashed in. Note that all observers agree on the order of events that are ordered in the underlying pomset. Formally, each observation is a linearization of the pomset or trace; different observers of the same trace may see different linearizations from the same experiment.

Pomsets and traces are histories of partial executions of a program. In both models, the denotation of a program is the set of all of its possible partial executions. That is, a *trace process* is simply a prefix-closed set of traces. Pomset processes are slightly more complex: for example, $\{a \parallel b\}$ and $\{a \parallel b, a ; b\}$ are indistinguishable. *Pomset processes* are sets of pomsets closed under prefixing and *augmentation*; that is, increasing the linearity of the partial order. It would be desirable that this semantics be *fully abstract*; that is, that two pomset processes be equal if and only if they are indistinguishable. In both cases, the scenario for distinguishing histories also distinguishes processes.

In summary, distinguishing pomsets requires two technical devices: a replacement operation which substitutes pomset processes for actions, and concurrent observation by many observers. Distinguishing traces is much simpler, requiring only a sequencing operation and one observer in the worst case. Though both models are understandable in terms of observations, it is mathematically and philosophically easier to understand the distinctions between traces than between pomsets.

4.1 Distinguishing Pomsets (Pratt and Plotkin)

It is easy to find pomsets which appear the same to a single observer who is simply watching the pomset; for example, $a ; a$ and $a \parallel a$ both produce two a's. Pratt and Plotkin (18) give an experimental scenario for distinguishing arbitrary pomsets. The scenario uses two tools: (1) a *refinement* operation, which has the effect of replacing each action a by a process P_a, and (2)

multiple observers, who may see different orders of independent events.[3]

If p is a pomset, a an action, and A is a set of actions, we define $p[a := A]$ to be the set of all pomsets which are like p except that each occurrence of an a is replaced (independently) by an element of A. For example,

$$\begin{aligned}(a\,;a)[a := \{a_1, a_2\}] &= \{(a_1\,;a_1), (a_1\,;a_2), (a_2\,;a_1), (a_2\,;a_2)\} \\ (a \parallel a)[a := \{a_1, a_2\}] &= \{a_1 \parallel a_1, a_1 \parallel a_2, a_2 \parallel a_2\}\end{aligned}$$

We define simultaneous substitution for many actions in the obvious way.

An observation of a set of pomsets is made by first nondeterministically choosing an element, and then watching linearizations of that element. The two sets of pomsets are still not distinguishable by a single observer; each set can produce any of the four strings a_1a_1, a_1a_2, a_2a_1, a_2a_2.

However, with two observers watching the same element, the two sets are distinguishable. In this scenario, the two observers may see different linearizations of the element. Two observers watching $(a\,;a)[a := \{a_1, a_2\}]$ will always see the same string. However, if the element of $(a \parallel a)[a := \{a_1, a_2\}]$ happens to be $a_1 \parallel a_2$, then one observer may see a_1a_2 and the other a_2a_1. This is a distinguishing observation.

Generalizing this method, Pratt and Plotkin give a scheme for distinguishing arbitrary pomsets:

Theorem 4.1 (Pratt and Plotkin(18))

1. *Let p and q be distinct pomsets. There exists a context $C[\cdot]$ (involving simultaneous replacement), and a multiple-observer observation λ (for some number of observers) such that one of $C[p]$ and $C[q]$ can yield λ and the other cannot.*

2. *Let P and Q be distinct pomset processes. Then P and Q may be distinguished by such a scenario.*

4.2 Distinguishing Traces

The situation for traces is considerably simpler: observing single linearizations suffices to distinguish traces. The precise statement of this fact requires some care; in particular, we must decide how visible independencies are. The details of this depend on our treatment of concurrent alphabets.

[3] There is a simpler scheme for distinguishing series-parallel pomsets, those built from actions by ; and $\parallel$.

First, we consider concurrent alphabets to be visible. For example, the black box may be several black boxes connected by cables; we might know that actions on different boxes were independent. Given this knowledge, distinguishing distinct traces is trivial, as they have no linearizations in common at all. Theorem 4.2 follows immediately from the definitions.

Theorem 4.2 *Let σ and τ be traces over the same concurrent alphabet. Then the following are equivalent:*

1. *σ and τ can be distinguished by linearizations; that is, there is some string x which is a linearization of precisely one of σ and τ.*

2. *σ and τ are disjoint; that is, they have* no *linearizations in common.*

3. *$\sigma \neq \tau$*

Proof:

1. $(1 \Rightarrow 2)$ Let σ, τ be traces over the same alphabet, and let $x \in \sigma$ such that $x \notin \tau$. Suppose there exists $y \in \sigma$ such that $y \in \tau$. Then $x \equiv_\iota y$, and hence $x \in \tau$, and we have reached a contradiction.

2. $(2 \Rightarrow 3)$ Let σ, τ be disjoint, then there exists $x \in \sigma$ such that $x \notin \tau$ (or symmetrically). As σ and τ are equivalence classes, they must be distinct.

3. $(3 \Rightarrow 1)$ Clear.

It is possible to structure the metaphor of processes as black boxes so that independence information is visible (*e.g.*, by giving the calculator some peripherals); however, for strict comparison with other systems, we should stay with the simpler case of no information about the concurrent alphabet.

Without knowing something about the concurrent alphabet, it is impossible to distinguish some distinct pairs of traces; *e.g.*, the empty trace looks empty over all alphabets. More generally, if a and b are never adjacent in any linearization of σ, then it is impossible to tell whether or not $a \iota b$.

So, we simply include a context which enters each pair of actions adjacent to each other. This context can be realized with sequential composition, or even CCS-style prefixing; it does not require the somewhat exotic operation of action refinement. We obtain the following:

Theorem 4.3 *If σ and τ are distinct traces (over possibly distinct concurrent alphabets) then there is a context C involving only sequential composition and a string x distinguishing them; that is, precisely one of $C[\sigma]$ and $C[\tau]$ can perform x.*

Proof: Let $\Sigma = \{a_1, \ldots, a_n\}$ be the set of actions in σ and τ. Let

$$p = a_1 a_1\, a_1 a_2\, a_1 a_3\, \ldots, a_1 a_n\, a_2 a_1, \ldots a_n a_n$$

be a string in which each pair of elements of Σ occurs consecutively. The experiment $C[X]$ consists of running a process which first executes p, and then runs X. If σ and τ are traces over different concurrent alphabets, then there is some permutation of p which can occur in the alphabet of one but not the other, and the two processes can be distinguished. Otherwise, Theorem 4.2 applies. ⏚

Essentially the same techniques may be used to distinguish trace processes:

Theorem 4.4 *Given two distinct trace processes, there is a context C involving only sequential composition and a string x distinguishing them.*

5 Consistent Completeness

The order-theoretic properties of traces under prefixing are significantly simpler than those of pomsets. In particular, traces (and even irreflexive pomsets) form a consistent-complete partial order (that is, if p_1 and p_2 have an upper bound, then they have a least upper bound), while admitting non-irreflexive pomsets breaks consistent completeness.

The latter can be shown by a counterexample. In Figure 6, we have $p_1, p_2 \leq p_3, p_4$. Furthermore, there are no processes between p_1, p_2 and p_3, p_4. So, p_1 and p_2 have no least upper bound.

In fact, we can prove that a large subclass of pomsets (pomsets with *no self-concurrency*) is consistently complete. This subclass includes irreflexive pomsets.

Definition 5.1 *The* level *of an event e of the pomset p (notation* $\mathsf{lev}(e)$*) is:*

$$max\{n \mid \exists e_1, e_2, \ldots, e_n.(e = e_n) \wedge (e_1 <_p e_2 <_p \ldots <_p e_n)\}$$

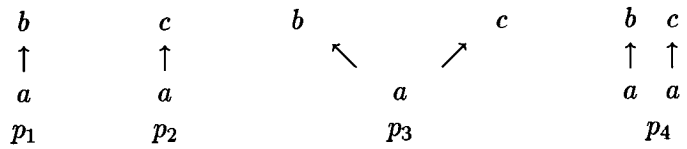

Figure 6: Failure of Consistent Completeness for Pomsets

Definition 5.2 *A pomset p is said to have* no self-concurrency *iff no two incomparable events at the same level have the same label,* i.e.*:*

$$\forall e, e' \in E_p.(e \perp e' \wedge \mathsf{lev}(e) = \mathsf{lev}(e')) \Rightarrow \mu(e) \neq \mu(e').$$

Note that the pomset p_4 in Figure 6 has self-concurrency. The following is an easy observation.

Lemma 5.3 *If p is an irreflexive pomset, then it has no self-concurrency.*

Proof: Easy. ⏚

Lemma 5.4 *Let $(P, \leq)$ be a poset with bottom element 0 such that for all $x \in P$ the set $\downarrow x = \{y \in P \mid y \leq x\}$ is finite. Then if every pair of elements $x, y \in P$ has a greatest lower bound $x \sqcap y \in P$ then $(P, \leq)$ is consistently complete.*

Proof:

1. Since every pair of elements has a greatest lower bound, the existence of greatest lower bounds of finite subsets X of P can be shown by induction on the cardinality of X.

2. Let $X \subseteq P$ be finite, and let $b \in P$ be a bound of X. Define $U_b \subseteq P$ to be the set of upper bounds of X no greater than b, *i.e.*:

$$U_b = \{y \in P \mid \forall x \in X.x \leq y \leq b\}$$

   Then U_b is non-empty (it contains b) and finite (because $\downarrow b$ is finite), and hence by part (1) U_b has a glb $\sqcap U_b$. We show that $u = \sqcap U_b = \sqcup X$.

   - $\forall x \in X, \forall b' \in U_b.x \leq b'$, so $\forall x \in X.x \leq u$, and hence u is an upper bound of X.

- Suppose $\forall x \in X.x \leq z$, some $z \in P$. By assumption there exists $d = z \sqcap b$. Since $\forall x \in X.x \leq z$ and $x \leq b$, we have $\forall x \in X.x \leq z \sqcap b = d$. So $d \in U_b$, and thus $\sqcap U_b \leq z$.

Lemma 5.5 *If p, q be pomsets with no self-concurrency, then their greatest lower bound $p \sqcap q$ exists.*

Proof:

It is convenient to work with canonical representatives of pomsets, designed to give the same names to events that might be the same in p and q. Suppose p is a pomset, and define a pomset $l(p)$ by:

$$\begin{array}{rcl} E_{l(p)} & = & \{\langle a, n\rangle \mid \exists e \in E_p.\mu(e) = a, \mathsf{lev}(e) = n\} \\ \langle a, i\rangle \leq_{l(p)} \langle b, j\rangle & \iff & \exists e_a, e_b.(\mu(e_a) = a \wedge \mu(e_b) = b \\ p & & \wedge \mathsf{lev}(e_a) = i \wedge \mathsf{lev}(e_b) = j \wedge e_a \leq_p e_b) \\ \mu(\langle a, i\rangle) & = & a \end{array}$$

If p has no self-concurrency, $p = l(p)$. (To see this, define $f : E_p \to E_{l(p)}$ by $f(e) = \langle a, i\rangle$ where $\mu(e) = a, \mathsf{lev}(e) = i$; it is easy to see that this is an order isomorphism). We thus abuse notation and choose $l(p)$ as our representative of p for the remainder of this proof.

If u and v are pomsets, we write $u \doteq v$ if $E_u = E_v$ (which, by the above convention, means $E_{l(u)} = E_{l(v)}$), and the identity map is a labelled partial order isomorphism between them. If p is a pomset and $e \in E_p$, then $\downarrow_p e$ is the minimal pomset prefix of p containing e.

It is straightforward to show that, if $s \leq p$, then

$$\forall e \in E_s.\downarrow_p e \doteq \downarrow_s e \tag{3}$$

Suppose p and q are non-self-concurrent pomsets. We define r as follows; r will be the infimum of p and q. We use the notation $X \restriction E$ for the restriction of the function or relation X to the set E.

$$\begin{array}{rcl} E_r & = & \left\{e \in E_p \cap E_q \mid \downarrow_p e \doteq \downarrow_q e\right\} \\ \leq_r & = & \leq_p \restriction E_r \\ \mu_r & = & \mu_p \restriction E_r \end{array}$$

Note that $\leq_r = \leq_q \restriction E_r$ as well; if $e_1 \leq_r e_2$, then $e_1 \leq_p e_2$. As the identity is an isomorphism between $\downarrow_p e_1$ and $\downarrow_q e_1$, we have $e_1 \leq_q e_2$ as desired.

1. We show $r \leq p, q$. It suffices to show that E_r is a downward-closed subset of E_p; E_q is symmetrical. Suppose that $e \in E_r$ and $E_p \ni e' \leq_p e$. As $\downarrow_p e \doteq \downarrow_q e$, $e' \in E_q$ and $\downarrow_p e' = \downarrow_q e'$. Hence $e' \in E_r$.

2. We show that, if $s \leq p, q$, then $s \leq r$. Clearly s has no auto-concurrency. Using $s = l(s)$, we see that $E_s \subseteq E_p \cap E_q$. By (3), we have $\downarrow_s e \doteq \downarrow_p e \doteq \downarrow_q e$. Hence

$$\begin{aligned} E_s &= \left\{ e \in E_s \cap E_p \cap E_q | \downarrow_s e \doteq \downarrow_p e \doteq \downarrow_q e \right\} \\ &\subseteq \left\{ e \in E_p \cap E_q | \downarrow_p e \doteq \downarrow_q e \right\} \\ &= E_r \end{aligned}$$

and

$$\begin{aligned} \leq_r &= \leq_p \restriction E_r \\ \leq_s &= \leq_p \restriction E_s \\ &= (\leq_p \restriction E_r) \restriction E_s \\ &= \leq_r \restriction E_s \end{aligned}$$

and clearly the labels are correct. Hence by construction $s \leq r$.

Thus r is the desired greatest lower bound. $\Box$

Figure 7 shows examples of greatest lower bounds for pomsets.

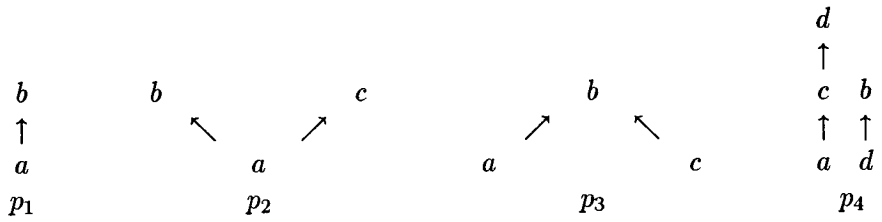

Figure 7: Examples of Greatest Lower Bounds for Pomsets: $p_1 = p_1 \sqcap p_2, a = p_1 \sqcap p_3, a = p_1 \sqcap p_4$.

The main theorem of this section now follows.

Theorem 5.6 *Finite pomsets with no self-concurrency are consistently complete.*

Proof: To prove consistent completeness for finite pomsets with no self-concurrency, first observe that for every pomset p the set $\{q \mid q \leq p\}$ is finite. By Lemma 5.5, every pair of pomsets with no self-concurrency has a greatest lower bound. The result follows directly from Lemma 5.4. ⏚

Corollary 5.7 *Irreflexive pomsets are consistently complete.*

Proof: Direct from Lemma 5.3 and Theorem 5.6. ⏚

By the order-isomorphism (Theorem 3.21), consistent completeness of Mazurkiewicz traces also follows from Theorem 5.6. For an alternative proof involving possibly infinite traces see *e.g.* (13).

Corollary 5.8 *Mazurkiewicz traces are consistently complete in trace prefix order* $\sqsubseteq$.

Proof: Direct from Theorem 5.6 and Theorem 3.21. ⏚

6 Operations

Execution histories are not used in isolation. They are frequently built from simple histories; for example, a system might be specified by saying that first it performs A, then it performs B and C in parallel, and finally it performs D. Pomsets are well-suited for this sort of specification; a wide variety of operations are available on pomsets (20). Central among these operations are concurrency $\|$ and sequencing ; already described. The algebra of these operations is developed in (7).

Neither operation is well-defined on Mazurkiewicz traces. Actions cannot be in parallel with themselves. For example, it is impossible for a daisywheel printer to print two a's at the same time; any two such events must be sequenced. Similarly, if a and b are independent, then they cannot happen sequentially without some form of communication. For example, let M and N be computers separated by an asynchronous network without a global clock. It is impossible to have first M, then N flash a light without some intervening communication in which M tells N that M has finished flashing

its light. This is an intuitive motivation for forbidding general concurrency and sequencing.

More precisely, irreflexivity conflicts with parallel composition: $a \parallel a$ is p_{indep}, the example of a non-irreflexive pomset of Figure 5. Similarly, independence conflicts with sequential composition, and $(a\,;b)\,;(a \parallel b)$ is p_{irr} of the same figure.

Mazurkiewicz traces use a single operation of concatenation which has aspects of both sequencing and concurrency, making actions concurrent when possible and sequencing otherwise. However, both sequencing and concurrency are given by concatenation of suitable strings. This can be understood by the addition of a type system. First, all work should be done in the same concurrent alphabet; it makes little sense to have a and b as letters on a printer in one part of the specification, and actions on different processors in another. We fix a concurrent alphabet for the remainder of this discussion; properly, it should be part of the type structure.

Definition 6.1

1. *A* type A *is a set of actions. A trace* σ *has type* A*, written* $\sigma : A$*, if all actions in* σ *are elements of* A*.*

2. *Two types are* concurrency compatible *iff for all* $a \in A$*,* $b \in B$ *we have* $a \iota b$*.*

3. *Two types are* sequencing compatible *iff for all* $a \in A$*,* $b \in B$ *we have* $\neg(a \iota b)$*.*

Intuitively, concurrency compatible types are alphabets of actions on separated machines (in particular, unlike CCS and CSP, traces do not synchronize on matching actions)[4], while sequencing compatible types are alphabets of actions on the same (or closely connected) machines.

Let $\sigma : A$ and $\tau : B$. If A and B are concurrency compatible, then the two traces occur on separate machines, and so it makes sense to execute them in parallel. In fact, $\sigma \cdot \tau$ represents σ and τ running in parallel. The corresponding fact holds for sequencing. Our connection with pomsets (in which concurrency and sequencing are defined) allows us to state this formally:

[4]Both pomsets and Mazurkiewicz traces have CCS or CSP style synchronization operations, which run sequences of actions in parallel synchronizing on shared actions. We expect similar results to hold for these operations.

Theorem 6.2 *Let $\sigma : A$ and $\tau : B$.*

1. *If A and B are concurrency compatible, then* $\mathsf{pom}(\sigma \cdot \tau) = \mathsf{pom}(\sigma) \parallel \mathsf{pom}(\tau)$.
2. *If A and B are sequencing compatible, then* $\mathsf{pom}(\sigma \cdot \tau) = \mathsf{pom}(\sigma) ; \mathsf{pom}(\tau)$.

Proof:

1. Let σ, τ be traces over concurrency compatible alphabets, and let $E_{\sigma \cdot \tau}$ denote the set of events (points) of $\mathsf{pom}(\sigma \cdot \tau)$, $E_{\sigma \| \tau}$ the set of events of $\mathsf{pom}(\sigma) \parallel \mathsf{pom}(\tau)$. We have $\sigma \sqsubseteq \sigma \cdot \tau$, and hence $E_\sigma \subseteq E_{\sigma \cdot \tau}$. Also, since the alphabets are concurrency compatible, we have $\tau \sqsubseteq \tau \cdot \sigma = \sigma \cdot \tau$, from which it follows $E_\tau \subseteq E_{\sigma \cdot \tau}$. It is easy to see that $E_{\sigma \cdot \tau} = E_\sigma \cup E_\tau = E_{\sigma \| \tau}$. Define $f : E_{\sigma \cdot \tau} \to E_{\sigma \| \tau}$ by:

   $$f([pa]_\iota) = [pa]_\iota$$

   It is easy to see that this is an order- and label-preserving isomorphism.

2. Similarly.

△

It remains to be seen if there is a clear connection between $\mathsf{pom}(\sigma \cdot \tau)$, $\mathsf{pom}(\sigma)$, and $\mathsf{pom}(\tau)$ in the case where A and B are neither concurrency nor sequencing compatible. We feel that a synchronization operation would have to be used.

7 Conclusion

Pomsets amount to a model of untyped true concurrency, in which arbitrary combinations of actions are possible. This generality gives it great power in modeling and specifying processes. Mazurkiewicz traces amount to a model of simply-typed true concurrency. In this model, some actions are pre-defined to be independent (*e.g.*, occurring on different machines) and others are dependent (*e.g.*, on the same machine). This additional structure imposes restrictions on which operations are possible (*e.g.*, arbitrary sequencing and concurrency are not possible, as they can be used to violate the independence relation), but these restrictions pay off in mathematical simplicity (exemplified by consistent completeness) and foundational simplicity (exemplified by the simpler experiments used to distinguish between

the two). In this paper, the proofs concerning traces tended to be simpler than those concerning pomsets. In particular, the induction scheme on traces was straightforward (as with strings, if $[s]_\iota \neq [\epsilon]_\iota$, then $[s]_\iota = [c]_\iota [s']_\iota$).

This suggests the following methods of usage. It seems likely that most situations will be simply typed, and hence can be described in terms of Mazurkiewicz traces. It is better to use traces whenever possible, as they are significantly simpler and have more structure. In a few situations, they will prove inadequate. Then, the work can be translated into the more powerful setting of pomsets, using the theory of Section 3.

7.1 Open Problems

Continuing this form of analysis with other restricted classes of pomsets would be helpful. For example, the classes of *series-parallel* and *codot* pomsets (7) and seems to be mathematically tractable, and may have advantages similar to Mazurkiewicz traces in other settings.

We have only discussed operations on execution histories in both models, as a basis for appropriate operations on process denotations (*i.e.* prefix-closed sets of traces or pomsets). To build a denotational framework for true concurrency semantics, it would be desirable to derive and relate suitable operations on pomset and trace processes. Ideally, these operations should be continuous, either in the metric sense or in the sense of Scott, and should model sequential and parallel composition, non-determinism and synchronization. The order-theoretic and metric properties of traces have been investigated in (13,12). Pomsets form a complete metric space (5), but a further analysis of the topological properties is necessary.

Another interesting problem is to provide a typed process calculus for which Mazurkiewicz traces are fully abstract.

8 Acknowledgments

The first author would like to thank Vicki Borah for terminological assistance. The second author would like to acknowledge the Amsterdam Concurrency Group at CWI, especially Jeroen Warmerdam, for helpful discussions on this and related subjects.

References

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

[2] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Sci.*, 53(2/3):225–241, 1987.

[3] B. Bloom and A. R. Meyer. Experimenting with process equivalence. In D. M. Kwiatkowska, editor, *Proceedings of the International BCS-FACS Workshop on Semantics for Concurrency*, pages 189–201, Leicester, U.K., July 1990.

[4] M. Chandy and L. Lamport. Finding global states of a distributed system. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[5] J. W. de Bakker and J. H. A. Warmerdam. Metric pomset semantics for a concurrent language with recursion. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lect. Notes in Computer Sci.*, pages 21–50. LITP Spring School on Theoretical Computer Science, Springer-Verlag, Apr. 1990.

[6] R. de Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Sci.*, 34(2/3):83–133, 1984.

[7] J. L. Gischer. Partial orders and the axiomatic theory of schuffle. Technical Report STAN-CS-84-1033, Stanford University, 1984.

[8] J. Grabowski. On partial languages. *Fundamenta Informaticae*, IV(2):427–498, 1981.

[9] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall, 1985.

[10] M. Kwiatkowska. *Fairness for Non-interleaving Concurrency.* PhD thesis, University of Leicester, 1989.

[11] M. Z. Kwiatkowska. Defining process fairness for non-interleaving concurrency. In K. Nori and Veni-Madhavan, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *Lecture Notes in Computer Science*, pages 286–300. Springer-Verlag, 1990.

[12] M. Z. Kwiatkowska. A metric for traces. *Information Processing Letters*, 35(3):129–135, 1990.

[13] M. Z. Kwiatkowska. On the domain of traces and sequential composition. In S. Abramsky and T. Maibaum, editors, *TAPSOFT'91*, volume 493 of *Lecture Notes in Computer Science*, pages 42–56. Springer-Verlag, 1991.

[14] A. Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI Report PB-78, Aarhus University, 1977.

[15] A. Mazurkiewicz. Basic notions of trace theory. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 112 of *Lect. Notes in Computer Sci.*, pages 25–34. Springer-Verlag, 1989.

[16] R. Milner. A modal characterisation of observable machine-behaviour. In E. Astesiano and C. Böhm, editors, *CAAP '81: Trees in Algebra and Programming, 6th Colloquium*, volume 112 of *Lect. Notes in Computer Sci.*, pages 25–34. Springer-Verlag, 1981.

[17] R. Milner. *Communication and Concurrency.* Prentice Hall International Series in Computer Science. Prentice Hall, New York, 1989.

[18] G. Plotkin and V. Pratt. The resolving power of multiple observers (extended abstract). (The final version has not yet appeared, to my knowledge), 1990.

[19] V. Pratt. The pomset model of parallel processes: Unifying the temporal and the spatial. In S. Brookes, A. Roscoe, and G. Winskel, editors, *Proc. CMU/SERC Workshop on Analysis of Concurrency*, volume 197 of *Lect. Notes in Computer Sci.* Springer-Verlag, 1985. LNCS 196.

[20] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[21] M. W. Shields. Elements of a theory of parallelism. (to be published).

[22] P. H. Starke. Traces and semiwords. In A. Skowron, editor, *Computation Theory*, volume 208 of *Lect. Notes in Computer Sci.*, pages 332–349. Springer-Verlag, 1985.

[23] E. Szpilrajn. Sur l'extension de l'ordre partiel. *Fund. Math.*, 16:386–389, 1930.

[24] K. Taylor and C. Critchlow. The inhibition spectrum and the achievement of causal consistency (extended abstract). In *Proceedings of the 9th SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '90)*, 1990. (to appear August 1990).