In proceedings of MFCS '93, LNCS n. 711, 1993

Deterministic Behavioural Models for Concurrency

Vladimiro Sassone[*] *Mogens Nielsen*[**]
Glynn Winskel[**]

[*]Dipartimento di Informatica, Università di Pisa, Italy
[**]Computer Science Department, Aarhus University, Denmark

Extended Abstract

This paper offers three candidates for a deterministic, noninterleaving, behaviour model which generalizes Hoare traces to the noninterleaving situation. The three models are all proved equivalent in the rather strong sense of being equivalent as categories. The models are: deterministic labelled event structures, generalized trace languages in which the independence relation is context-dependent, and deterministic languages of pomsets.

Introduction

Models for concurrency can be classified according to whether they can represent the structure of *systems* or just their *behaviours* (*Behaviour or System model*); whether they can faithfully take into account the difference between *concurrency* and *nondeterminism* (*Interleaving or Noninterleaving model*); and, finally, whether they can represent the *branching structure* of processes, i.e., the points in which choices are taken, or not (*Linear or Branching Time model*).

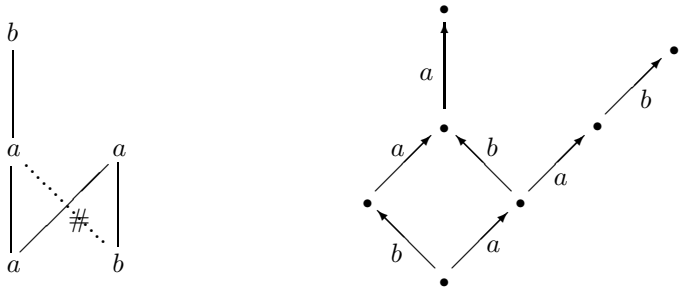
In [9], the authors studied a range of models based on such a classification. The classification has the shape of a *cube* whose vertices are *categories* of models—corresponding to the eight classes of models obtained by varying the parameters above—and whose edges establish formal relationships between such categories. More precisely, the edges of the cube are special forms of *adjunctions*, namely *reflections* or *coreflections*, which express translations between models.

Generally speaking, the model chosen to represent a class is a canonical and universally accepted representative of that class. For the behavioural models they are Hoare languages [3] for interleaving, linear-time models, synchronization trees [12] for interleaving, branching-time models, and labelled event structures [13] for noninterleaving, branching-time models. However, for the class of noninterleaving, linear-time models, there does not, at present, seem to be an obvious choice of a corresponding canonical model.

The choice taken in [9] is deterministic labelled event structures, i.e. labelled event structures where the enabling relation between configurations and events is deterministic in the sense that whenever two events with the same label are enabled at a common configuration they are the same. The following is an example of such an event structure,

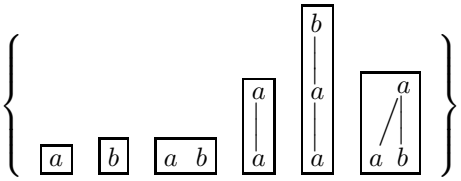
[*]This author has been supported by a grant of the Danish Research Academy.

together with its domain of configurations.



The choice of deterministic labelled event structures is based, by analogy, on the observation that Hoare trace languages may be viewed as deterministic synchronization trees, and that labelled event structures are a canonical generalization of synchronization trees within noninterleaving models. In this paper we investigate the relationship between this model and two of the most-studied, noninterleaving generalizations of Hoare languages in the literature: the pomsets of Pratt [8], and the traces of Mazurkiewicz [6].

Pomsets, an acronym for *p*artial *o*rdered *m*ulti*sets*, are labelled partial ordered sets. A noninterleaving representation of a system can be readily obtained by means of pomsets simply by considering the (multiset of) labels occurring in the run ordered by the *causal dependency* relation inherited from the events. The system itself is then represented by a set of pomsets. For instance, the labelled event structure given in the example discussed above can be represented by the following set of pomsets.



A simple but conceptually relevant observation about pomsets is that strings can be thought of as a particular kind of pomsets, namely those pomsets which are *finite* and *linearly* ordered. In other words, a pomset $\boxed{a_1 < a_2 < \cdots < a_n}$ represents the string $a_1 a_2 \cdots a_n$. On the other side of such correspondence, we can think of (finite) pomsets as a generalization of the notion of word (string) obtained by relaxing the constraint which imposes that the symbols in a word be linearly ordered. This is why in the literature pomsets have also appeared under the name *partial words* [2]. The analogy between pomsets and strings can be pursued to the point of defining languages of partial words, called *partial languages*, as prefix-closed—for a suitable extension of this concept to pomsets—sets of pomsets on a given alphabet of labels.

Since our purpose is to study linear-time models which are deterministic, we shall consider only pomsets without *autoconcurrency*, i.e., pomsets such that all the elements carrying the same label are linearly ordered. Following [11], we shall refer to this kind of pomsets as *semiwords* and to the corresponding languages as *semilanguages*. We shall identify a category **dSL** of deterministic semilanguages equivalent to the category of deterministic labelled event structures. Although pomsets have been studied extensively (see e.g. [8, 1, 2]), there are few previous results about formal relationships of pomsets with other models for concurrency.

Mazurkiewicz trace languages [6] are defined on an alphabet L together with a symmetric irreflexive binary relation I on L, called the *independence relation*. The relation I induces an equivalence on the strings of L^* which is generated by the simple rule

$$\alpha ab\beta \simeq \alpha ba\beta \quad \text{if} \quad a\ I\ b,$$

where $\alpha, \beta \in L^*$ and $a, b \in L$. A trace language is simply a subset M of L^* which is prefix-closed and $\simeq$-closed, i.e., $\alpha \in M$ and $\alpha \simeq \beta$ implies $\beta \in M$. It represents a system by representing all its possible behaviours as the sequences of (occurrences of) events it can perform. Since the independence relation can be taken to indicate the events which are *concurrent* to each other, the relation $\simeq$ does nothing but relate runs of the systems which differ only in the order in which independent events occur.

However, Mazurkiewicz trace languages are too abstract to describe faithfully labelled event structures. Consider for instance the labelled event structure shown earlier. Clearly, any trace language with alphabet $\{a, b\}$ able to describe such a labelled event structure must be such that $ab \simeq ba$. However, it cannot be such that $aba \simeq aab$. Thus, we are forced to move from the well-known model of trace languages. We shall introduce here a new notion of *generalized Mazurkiewicz trace language*, in which the independence relation is *context-dependent*. For instance, the event structure shown in the above picture will be represented by a trace language in which a is independent from b *at* ϵ, i.e., after the empty string, in symbols $a\ I_\epsilon\ b$, but a is *not* independent from b *at* a, i.e., after the string a has appeared, in symbols $a\ \not{I}_a\ b$. In particular, we shall present a category **GTL** of generalized trace languages which is equivalent to the category **dLES** of deterministic labelled event structures. We remark that a similar idea of generalizing Mazurkiewicz trace languages has been considered also in [4].

Summing up, we present the chain of equivalences

$$\mathbf{dSL} \simeq \mathbf{dLES} \simeq \mathbf{GTL}$$

which, besides identifying models which can replace **dLES** in our classification, also introduce interesting *deterministic behavioural models* for concurrency and formalizes their mutual relationships. This being an extended abstract, all the proofs are omitted; however, all the translations between models are presented in full. The reader interested in the complete treatment is referred to [10].

1 Preliminaries

One of the most studied noninterleaving models for concurrency is that of *event structures* [7, 12]. Their first class objects are events, assumed to be the *atomic computational steps*, which are related to each other by cause/effect and conflict relationships.

Definition 1.1 (*Labelled Event Structures*)
A labelled event structure is a structure $ES = (E, \#, \leq, \ell, L)$ *consisting of a set of events* E *partially ordered by* $\leq$*; a symmetric, irreflexive relation* $\# \subseteq E \times E$*, the conflict relation, such that*

$$\{\, e' \in E \mid e' \leq e \,\} \text{ is finite for each } e \in E,$$

$$e \# e' \leq e'' \text{ implies } e \# e'' \text{ for each } e, e', e'' \in E;$$

a set of labels L and a labelling function $\ell\colon E \to L$. For an event $e \in E$, define $\lfloor e \rfloor = \{\, e' \in E \mid e' \leq e \,\}$. Moreover, we write $\Cup$ for the relation $\# \cup \{\, (e,e) \mid e \in E_{ES} \}$.
A labelled event structure morphism from ES_0 to ES_1 is a pair of partial functions (η, λ), where $\eta\colon E_{ES_0} \rightharpoonup E_{ES_1}$ and $\lambda\colon L_{ES_0} \rightharpoonup L_{ES_1}$ are such that

$$(i)\ \ \lfloor \eta(e) \rfloor \subseteq \eta(\lfloor e \rfloor), \quad (ii)\ \ \eta(e) \Cup \eta(e') \Rightarrow e \Cup e', \quad (iii)\ \ \lambda \circ \ell_{ES_0} = \ell_{ES_1} \circ \eta.$$

Taking composition to be the componentwise composition of partial functions, defines the category **LES** *of labelled event structures.*

The computational intuition behind event structures is very simple: an event e can occur when all its *causes*, i.e., $\lfloor e \rfloor \setminus \{e\}$, have occurred and no event which it is in conflict with has already occurred. This is formalized by the following notion of *configuration*.

Definition 1.2 (*Configurations*)
Given a labelled event structure ES, define the configurations of ES to be those subsets $c \subseteq E_{ES}$ which are

$$\textit{Conflict Free:} \quad \forall e_1, e_2 \in c,\ \textit{not}\ e_1 \# e_2$$
$$\textit{Left Closed:} \quad \forall e \in c\ \forall e' \leq e,\ e' \in c$$

Let $\mathcal{L}(ES)$ denote the set of configurations of ES. We say that event e is enabled at a configuration c, in symbols $c \vdash e$, if (i) $e \notin c$; (ii) $\lfloor e \rfloor \setminus \{e\} \subseteq c$; and (iii) $e' \in E_{ES}$ and $e' \# e$ implies $e' \notin c$.

Definition 1.3 (*Deterministic Event Structures*)
A labelled event structure ES is deterministic if and only if for any $c \in \mathcal{L}(ES)$, and for any pair of events $e, e' \in E_{ES}$, whenever $c \vdash e$, $c \vdash e'$ and $\ell(e) = \ell(e')$, then $e = e'$.
This defines the category **dLES** *as a full subcategory of* **LES**.

Configurations of event structures may be viewed as *labelled partial orders* on L, i.e., as triples $(E, \leq, \ell)$, where E is a set, $\leq \subseteq E^2$ a partial order relation; and $\ell\colon E \to L$ is a *labelling* function. We say that a labelled partial order $(E, \leq, \ell)$ is *finite* if E is so.

Definition 1.4 (*Partial Words*)
A partial word on L is an isomorphism class of finite labelled partial orders. Given a finite labelled partial order p we shall denote with $[\![p]\!]$ the partial word which contains p. We shall also say that p represents the partial word $[\![p]\!]$.

A semiword is a partial word which does not exhibit autoconcurrency, i.e., such that all its subsets consisting of elements carrying the same label are linearly ordered. This is a strong simplification. Indeed, given a labelled partial order p representing a semiword on L and any label $a \in L$, such hypothesis allows us to talk *unequivocally* of the first element labelled a, of the second element labelled a, ..., the n-th element labelled a. In other words, we can represent p unequivocally as a (strict) partial order whose elements are pairs in $L \times \omega$, (a, i) representing the i-th element carrying label a. Thus, we are led to the following definition, where for n a natural number, $[n]$ denote the initial segment of length n of $\omega \setminus \{0\}$, i.e., $[n] = \{1, \ldots, n\}$.

DEFINITION 1.5 (*Semiwords*)
A (canonical representative of a) semiword on an alphabet L is a pair $x = (A_x, <_x)$ where

- $A_x = \bigcup_{a \in L} \Big(\{a\} \times [n_a^x] \Big)$, *for some $n_a^x \in \omega$, and A_x is finite;*
- $<_x$ *is a transitive, irreflexive, binary relation on A_x such that*

$$(a,i) <_x (a,j) \quad \textit{if and only if} \quad i < j,$$

where $<$ is the usual (strict) ordering on natural numbers.

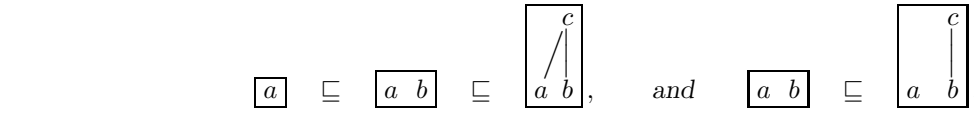
The semiword represented by x is $\Big[\!\Big[(A_x, \leq, \ell) \Big]\!\Big]$, where $(a,i) \leq (b,j)$ if and only if $(a,i) <_x (b,j)$ or $(a,i) = (b,j)$, and $\ell\Big((a,i)\Big) = a$. However, exploiting in full the existence of such an easy representation, from now on, we shall make no distinction between x and the semiword which it represents. In particular, as already stressed in Definition 1.5, with abuse of language, we shall refer to x as a semiword. The set of semiwords on L will be indicated by $SW(L)$. The usual set of words (strings) on L is (isomorphic to) the subset of $SW(L)$ consisting of semiwords with *total* ordering.

A standard ordering used on words is the prefix order $\sqsubseteq$, which relates α and β if and only if α is an initial segment of β. Such idea is easily extended to semiwords in order to define a prefix order $\sqsubseteq \subseteq SW(L) \times SW(L)$. Consider x and y in $SW(L)$. Following the intuition, for x to be a prefix of y, it is necessary that the elements of A_x are contained also in A_y with the same ordering. Moreover, since new elements can be added in A_y only "on the top" of A_x, no element in $A_y \setminus A_x$ may be less than an element of A_x. This is formalized by saying

$$\begin{array}{lll} x \sqsubseteq y \quad \text{if and only if} \quad A_x \subseteq A_y & \text{and} & <_x = <_y \cap A_x^2 \\ & \text{and} & <_y \cap ((A_y \setminus A_x) \times A_x) = \varnothing. \end{array}$$

It is quickly realized that $\sqsubseteq$ is a partial order on $SW(L)$ and that it coincides with the usual prefix ordering on words.

EXAMPLE 1.6 (*Prefix Ordering*)
As a few examples of the prefix ordering of semiwords, it is



However, it is neither the case that

We shall use $Pref(x)$ to denote the set $\{y \in SW(L) \mid y \sqsubset x\}$ of *proper prefixes* of x. The set of maximal elements in x will be denoted by $Max(x)$. Semiwords with

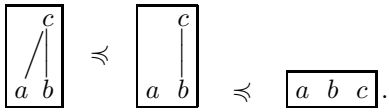
a maximum element play a key role in our development. For reasons that will be clear later, we shall refer to them as to *events*.

Another important ordering is usually defined on semiwords: the *"smoother than"* order, which takes into account that a semiword can be extended just by relaxing its ordering. More precisely, x is smoother than y, in symbols $x \preccurlyeq y$, if x imposes more order contraints on the elements of y. Formally,

$$x \preccurlyeq y \qquad \text{if and only if} \qquad A_x = A_y \quad \text{and} \quad <_x \supseteq <_y.$$

It is easy to see that $\preccurlyeq \subseteq SW(L) \times SW(L)$ is a partial order. In the following, we shall use $Smooth(x)$ to denote the set of *smoothings* of x, i.e., the set $\{y \in SW(L) \mid y \preccurlyeq x\}$.

Example 1.7 (*Smoother than Ordering*)
The following few easy situations exemplify the smoother than ordering of semiwords.



On the other hand, neither

2 Semilanguages and Event Structures

Semilanguages are a straightforward generalization of Hoare languages to prefix-closed subsets of $SW(L)$.

Definition 2.1 (*SemiLanguages*)
A semilanguage is a pair (SW, L), where L is an alphabet and SW is a set of semiwords on L which is

$$\begin{array}{ll} \textit{Prefix closed:} & y \in SW \text{ and } x \sqsubseteq y \quad \textit{implies} \quad x \in SW; \\ \textit{Coherent:} & Pref(x) \subseteq SW \text{ and } |Max(x)| > 2 \quad \textit{implies} \quad x \in SW. \end{array}$$

Semilanguage (SW, L) is deterministic if

$$x, y \in SW \text{ and } Smooth(x) \cap Smooth(y) \neq \varnothing \quad \textit{implies} \quad x = y.$$

In order to fully understand this definition, we need to appeal to the intended meaning of semilanguages. A semiword in a semilanguage describes a (partial) run of a system in terms of the observable properties (labels) of the events which have occurred, together with the causal relationships which rule their interactions. Thus, the *prefix closedness* clause captures exactly the intuitive fact that any *initial segment* of a (partial) computation is itself a (partial) computation of the system.

In this view, the *coherence* axiom can be interpreted as follows. Suppose that there is a semiword x whose proper prefixes are in the language, i.e., they are runs of the system,

and suppose that $|Max(x)| > 2$. This means that, given any pair of maximal elements in x, there is a computation of the system in which the corresponding events have both occurred. Then, in this case, the coherence axiom asks for x to be a possible computation of the system, as well. In other words, we can look at coherence as to the axiom which forces a set of events to be conflict free if it is *pairwise* conflict free, as in [7] for *prime event structures* and in [6] for *proper trace languages*.

To conclude our discussion about Definition 2.1, let us analyze the notion of *determinism*. Remembering our interpretation of semiwords as runs of a system, it is easy to realize how the existence of distinct x and y such that $Smooth(x) \cap Smooth(y) \neq \varnothing$ would imply nondeterminism. In fact, if there were two different runs with a common linearization, then there would be two different computations exhibiting the same observable behaviour, i.e., in other words, two *non equivalent* sequences of events with the same strings of labels.

Also the notion of morphism of semilanguages can be derived smoothly as an extension of the existing one for Hoare languages.

Any $\lambda\colon L_0 \rightharpoonup L_1$ determines a partial function $\hat{\lambda}\colon SW(L_0) \rightharpoonup SW(L_1)$ which maps x to its *relabelling* through λ, if this represents a semiword, and is undefined otherwise. Consider now semilanguages (SW_0, L_0) and (SW_1, L_1), and suppose for $x \in SW_0$ that $\hat{\lambda}$ is defined on x. Although one could be tempted to ask that $\hat{\lambda}(x)$ be a semiword in SW_1, this would be by far too strong a requirement. In fact, since in $\hat{\lambda}(x)$ the order $<_x$ is strictly preserved, morphisms would always strictly preserve causal dependency, and this would be out of tune with the existing notion of morphism for event structures, in which sequential tasks can be simulated by *"more concurrent"* ones. Fortunately enough, we have an easy way to ask for the existence of a more concurrent version of $\hat{\lambda}(x)$ in SW_1. It consists of asking that $\hat{\lambda}(x)$ be a smoothing of some semiword in SW_1.

Definition 2.2 (*Semilanguage Morphisms*)
Given the semilanguages (SW_0, L_0) and (SW_1, L_1), a partial function $\lambda\colon L_0 \rightharpoonup L_1$ is a morphism $\lambda\colon (SW_0, L_0) \to (SW_1, L_1)$ if [1]

$$\forall\, x \in SW_0 \qquad \hat{\lambda}{\downarrow}x \quad \textit{and} \quad \hat{\lambda}(x) \in Smooth(SW_1).$$

It is worth observing that, if (SW_1, L_1) is *deterministic*, there can be *at most one* semiword in SW_1, say x_λ, such that $\hat{\lambda}(x) \in Smooth(x_\lambda)$. In this case, we can think of $\lambda\colon (SW_0, L_0) \to (SW_1, L_1)$ as mapping x to x_λ.

Example 2.3
Given $L_0 = \{a, b\}$ and $L_1 = \{c, d\}$, consider the deterministic semilanguages below.

$$SW_0 \;=\; \left\{ \varnothing \quad \boxed{a} \quad \boxed{b} \quad \boxed{\begin{matrix} a \\ | \\ b \end{matrix}} \right\} \qquad SW_1 \;=\; \left\{ \varnothing \quad \boxed{c} \quad \boxed{d} \quad \boxed{c \;\; d} \right\}.$$

[1] Here, and in the following, we use $f{\downarrow}x$ to mean that a partial function f is defined on argument x.

Then, the function λ which maps a to c and b to d is a morphism from (SW_0, L_0) to (SW_1, L_1). For instance,

$$\hat{\lambda}\left(\boxed{\begin{array}{c} a \\ | \\ b \end{array}} \right) \;=\; \boxed{\begin{array}{c} c \\ | \\ d \end{array}} \;\preccurlyeq\; \boxed{c \;\; d}.$$

Observe that the function $\lambda' \colon L_0 \to L_1$ which sends both a and b to c is not a morphism since $\hat{\lambda}$ applied to $\boxed{b < a}$ gives $\boxed{c < c}$ which is not the smoothing of any semiword in SW_1, while $\lambda'' \colon L_1 \to L_0$ which sends both c and d to a is not a morphism from (SW_1, L_1) to (SW_0, L_0) since $\hat{\lambda}$ is undefined on $\boxed{c \;\; d}$.

It can be shown that semilanguages and their morphisms, with composition that of partial functions, form a category whose full subcategory consisting of deterministic semilanguages will be denoted by **dSL**. In the following, we shall define *translation* functors between **dLES** and **dSL**.

Given a deterministic semilanguage (SW, L) define $dsl.dles\Big((SW, L)\Big)$ to be the structure $(E, \leq, \#, \ell, L)$, where

- $E = \Big\{ e \;\Big|\; e \in SW,\ e \text{ is an } \textit{event}, \text{ i.e., } e \text{ has a maximum element} \Big\}$;
- $\leq \;= \sqsubseteq \cap E^2$;
- $\# = \Big\{ (e, e') \in E^2 \;\Big|\; e \text{ and } e' \text{ are incompatible wrt } \sqsubseteq \Big\}$;
- $\ell(e)$ is the label of the maximum element of e.

THEOREM 2.4
$dsl.dles\Big((SW, L)\Big)$ is a deterministic labelled event structure.

Consider now a deterministic labelled event structure $DES = (E, \leq, \#, \ell, L)$. Define $dles.dsl(DES)$ to be the structure (SW, L), where

$$SW = \Big\{ \big[\!\big[(c, \leq \cap\, c^2, \ell_{|c}) \big]\!\big] \;\Big|\; c \text{ is a } \textit{finite} \text{ configuration of } DES \Big\}.$$

THEOREM 2.5
$dles.dsl(DES)$ is a deterministic semilanguage.

It can be shown that *dsl.dles* and *dles.dsl* extend to functors which when composed with each other yield functors naturally isomorphic to identity functors. In other words, they form an *adjoint equivalence* [5, chap. III, pg. 91], i.e., an adjunction which is both a *reflection* and a *coreflection*. It is worthwhile noticing that this implies that the mappings *dsl.dles* and *dles.dsl* constitute a bijection between deterministic semilanguages and isomorphism classes of deterministic labelled event structures—isomorphism being identity up to the names of events.

THEOREM 2.6
The categories **dSL** *and* **dLES** *are equivalent.*

In fact, dropping the axiom of coherence in Definition 2.1 we get semilanguages equivalent to labelled *stable event structures* [12].

3 Trace Languages and Event Structures

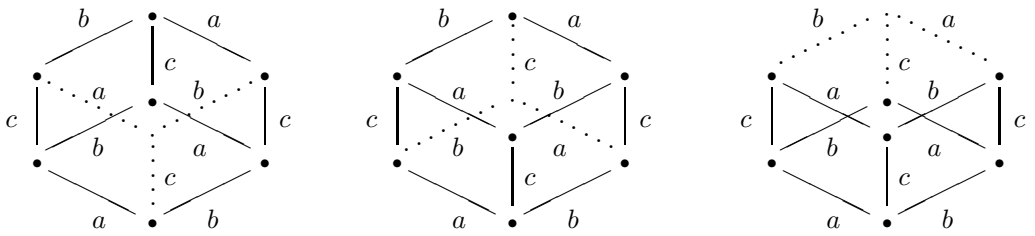
Generalized trace languages extend trace languages by considering an independence relation which may vary while the computation is progressing. Of course, we need a few axioms to guarantee the consistency of such an extension.

Definition 3.1 (*Generalized Trace Languages*)
A generalized trace language is a triple (M, I, L), where L is an alphabet, $M \subseteq L^*$ is a prefix-closed and $\simeq$-closed set of strings, $I\colon M \to 2^{L\times L}$ is a function which associates to each $s \in M$ a symmetric and irreflexive relation $I_s \subseteq L \times L$, such that

$$\begin{array}{lll} I \text{ is consistent:} & s \simeq s' \quad \text{implies} \quad I_s = I_{s'}; \\ M \text{ is } I\text{-closed:} & a\ I_s\ b \quad \text{implies} \quad sab \in M; \\ I \text{ is coherent:} & (i) \quad a\ I_s\ b \quad \text{and} \quad a\ I_{sb}\ c \quad \text{and} \quad c\ I_{sa}\ b \quad \text{implies} \quad a\ I_s\ c, \\ & (ii) \quad a\ I_s\ c \quad \text{and} \quad c\ I_s\ b \quad \text{implies} \quad (a\ I_s\ b \text{ if and only if } a\ I_{sc}\ b); \end{array}$$

where $\simeq$ is the least equivalence relation on L^* such that $sabu \simeq sbau$ if $a\ I_s\ b$.

As in the case of trace languages, we have an equivalence relation $\simeq$ which equates those strings representing the same computation. Thus, I must be consistent in the sense that it must associate the same independence relation to $\simeq$-equivalent strings. In order to understand the last two axioms, the following picture shows in terms of computations ordered by prefix the situations which those axioms forbid. There, the dots represent computations, the labelled edges represent the prefix ordering, and the dotted lines represent the computations forced in M by the axioms.



(A) (B) (C)

It is easy to see that axiom (i) rules out the situation described by just the solid lines in (A)—impossible for stable event structures, while axiom (ii) eliminates cases (B)—which is beyond the descriptive power of *general event structures* [12] and (C)—impossible for event structures with *binary* conflict. They narrow down to those orderings of computations arising from prime event structures.

Definition 3.2 (*Generalized Trace Language Morphisms*)
Given the generalized trace languages (M, I, L) and (M', I', L'), a partial function $\lambda\colon L \rightharpoonup L'$ is a morphism $\lambda\colon (M, I, L) \to (M', I', L')$ if

$$\begin{array}{lll} \lambda \text{ preserves words:} & s \in M \quad \text{implies} \quad \lambda^*(s) \in M'; \\ \lambda \text{ respects independence:} & a\ I_s\ b \quad \text{and} \quad \lambda{\downarrow}a,\ \lambda{\downarrow}b \quad \text{implies} \quad \lambda(a)\ I'_{\lambda^*(s)}\ \lambda(b); \end{array}$$

where λ^* *is inductively defined by* $\lambda^*(\epsilon) = \epsilon$ *and* $\lambda^*(sa) = \begin{cases} \lambda^*(s)\lambda(a) & \text{if } \lambda{\downarrow}a \\ \lambda^*(s) & \text{otherwise.} \end{cases}$

Generalized trace languages and their morphisms, under the usual composition of partial functions, form the category **GTL**.

A derived notion of event in generalized trace languages can be captured by the relation $\sim$ defined as the least equivalence such that

$$a\ I_s\ b \quad \text{implies} \quad sa \sim sba \qquad \text{and} \qquad s \simeq s' \quad \text{implies} \quad sa \sim s'a.$$

The events occurring in $s \in M$, denoted by $Ev(s)$, are the $\sim$-classes a representative of which occurs as a non empty prefix of s, i.e., $\Big\{ [u]_\sim \;\Big|\; u \text{ is a non empty prefix of } s \Big\}$. It can be shown that $s \simeq s'$ if and only if $Ev(s) = Ev(s')$. Extending the notation, we shall write $Ev(M)$ to denote the events of (M, I, L), i.e., the $\sim$-equivalence classes of non empty strings in M.

Now, given a generalized trace language (M, I, L) define $gtl.dles\Big((M, I, L)\Big)$ to be the structure $(Ev(M), \leq, \#, \ell, L)$, where

- $[s]_\sim \leq [s']_\sim$ if and only if $\forall u \in M,\ [s']_\sim \in Ev(u)$ implies $[s]_\sim \in Ev(u)$;
- $[s]_\sim \ \# \ [s']_\sim$ if and only if $\forall u \in M,\ [s]_\sim \in Ev(u)$ implies $[s']_\sim \notin Ev(u)$;
- $\ell\Big([s]_\sim\Big) = a$ if and only if $s = s'a$.

THEOREM 3.3
$gtl.dles\Big((M, I, L)\Big)$ is a deterministic labelled event structure.

On the other hand, in order to define a generalized trace language from a deterministic labelled event structure $DES = (E, \leq, \#, \ell, L)$, consider

$$M = \Big\{ \ell^*(e_1 \cdots e_n) \;\Big|\; \{e_1, \ldots, e_n\} \subseteq E \quad \text{and} \quad \{e_1, \ldots, e_{i-1}\} \vdash e_i,\ i = 1, \ldots, n \Big\}.$$

Since DES is deterministic, any $s \in M$ identifies unequivocally a string of events $Sec(s) = e_1 \cdots e_n \in E^*$ such that $\{e_1, \ldots, e_{i-1}\} \vdash e_i,\ i = 1, \ldots, n$, and $\ell^*(e_1 \cdots e_n) = s$. Now, for any $s \in M$, take $I_s = \Big\{ (a, b) \;\Big|\; sab \in M,\ Sec(sab) = xe_0e_1 \text{ and } e_0\ co\ e_1 \Big\}$. Then, define (M, I, L) to be $dles.gtl(DES)$.

THEOREM 3.4
$dles.gtl(DES)$ is a generalized trace language.

As in the case treated in the previous section, *dles.gtl* and *gtl.dles* extend to functors between **GTL** and **dLES** which form an adjoint equivalence. Such an equivalence restricts to an isomorphism of generalized trace languages and isomorphism classes of deterministic labelled event structures.

THEOREM 3.5
Categories **GTL** *and* **dLES** *are equivalent.*

The result extends to labelled *stable event structures* by dropping the *'only if'* implication in part *(ii)* of the coherence axiom of Definition 3.1. Of course, it follows from Theorem 2.6 and Theorem 3.5 that **dSL** and **GTL** are equivalent. In the full paper [10], we also define direct translations between such categories.

References

[1] J. Gischer. The Equational Theory of Pomsets. *Theoretical Computer Science*, n. 61, pp. 199–224, 1988.

[2] J. Grabowski. On Partial Languages. *Fundamenta Informaticae*, n. 4, pp. 428–498, 1981.

[3] C.A.R. Hoare. *Communicating Sequential Processes*. Englewood Cliffs, 1985.

[4] P.W. Hoogers, H.C.M Kleijn, and P.S. Thiagarajan. A Trace Semantics for Petri Nets. In Proceedings of *ICALP '92*, LNCS, n. 623, pp. 595–604, Springer Verlag, 1992.

[5] S. MacLane. *Categories for the Working Mathematician*. GTM, Springer-Verlag, 1971.

[6] A. Mazurkiewicz. Basic Notions of Trace Theory. In *lecture notes for the REX summer-school in temporal logic*, LNCS, n. 354, pp. 285–363, Springer-Verlag, 1988.

[7] M. Nielsen, G. Plotkin, and G. Winskel. *Petri nets, Event Structures and Domains, part 1*. Theoretical Computer Science, n. 13, pp. 85–108, 1981.

[8] V. Pratt. Modeling Concurrency with Partial Orders. *International Journal of Parallel Processing*, n. 15, pp. 33–71, 1986.

[9] V. Sassone, M. Nielsen, and G. Winskel. *A Classification of Models for Concurrency*. To appear as Technical Report Daimi, Computer Science Department, Aarhus University, 1993. Extended abstract to appear in Proceedings of *CONCUR '93*.

[10] V. Sassone, M. Nielsen, and G. Winskel. *Deterministic Behavioural Models for Concurrency*. To appear as Technical Report Daimi, Computer Science Department, Aarhus University, 1993.

[11] P. Starke. Traces and Semiwords. LNCS, n. 208, pp. 332–349, Springer-Verlag, 1985.

[12] G. Winskel. Event Structures. In *Advances in Petri nets*, LNCS, n. 255, pp. 325–392, Springer-Verlag, 1987.

[13] G. Winskel, and M. Nielsen. Models for Concurrency. To appear in the *Handbook of Logic in Computer Science*. A draft appears as DAIMI PB 429, 1992.