Math. Struct. in Comp. Science (1999), *vol.* 9, *pp.* 687–718. Printed in the United Kingdom

Observational trees as models for concurrency

STEFANO KASANGIAN[†] and ANNA LABELLA[‡]

[†] *Dip. di Matematica, Università di Milano, Italy.*
[‡] *Dip. di Scienze dell'Informazione, Università di Roma 'La Sapienza', Italy*

Received 30 June 1995; revised 10 April 1999

Given an automaton, its behaviour can be modelled as the sets of strings over an alphabet A that can be accepted from any of its states. When considering concurrent systems, we can see a concurrent agent as an automaton, where non-determinism derives from the fact that its states can offer a different behaviour at different moments in time. Non-deterministic computations between a pair of states can then no longer be described as a 'set' of strings in a free monoid. Consequently, between two states we will have a *labelled structured set of computations*, where the structure describes the possibility of two computations parting from each other while maintaining the same observable steps. In this paper, we shall consider different kinds of observation domains and related structured sets of computations. Structured sets of computations will be organised as a category of generalised trees built over a meet-semilattice monoid formalizing the observation domain. Theorems allowing us to introduce the usual concurrency operators in the models and relating different models will then be obtained by first considering ordinary functors (on and between the observation domains), and then lifting them to the categories of structured sets of computations.

1. Introduction

Parallel processes and distributed computations can be described and scrutinized from different viewpoints. Indeed, several approaches, based on different intuitions about the relevant aspects of concurrent systems, have been recommended to describe their behaviour. Often one of the distinguishing features of concurrency models is the stress they put on causality, co-occurrency and non-determinism. They range from *synchronization trees* (see, for example, Milner (1983) and Milner (1989)), where the stress is on the concepts of non-determinism and communication while causality is ignored and concurrency is rendered as non-deterministic interleaving, to Petri Nets (Reisig 1985), where concurrency, causality and independence are the basic building blocks. In between these two extremes, many other non-interleaving models have been introduced, among these we would like to mention Trace Theory (Mazurkiewicz 1987), Event Structures (Winskel 1987), Labelled Event Structures (Castellani *et al.* 1983), Concurrent Histories (Degano and Montanari 1987) and Pomsets (Pratt 1986). As the number of different models has increased, there have been correspndingly many attempts at comparing them. We refer the interested reader to Winskel *et al.* (1995), for a comprehensive exposition.

Non-determinism is a key notion in all the models mentioned above, and it is frequently

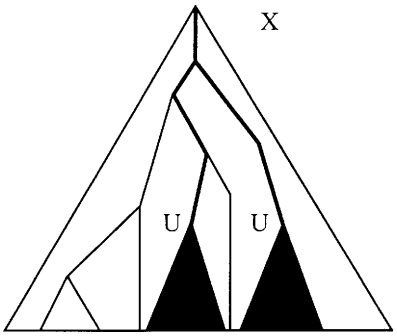
obtained by structuring the set of computations as trees and labelling them using a suitable alphabet of visible actions. The aim of this paper is to provide a general categorical account of tree-like labelled sets of computations, in order to capture many of the proposed concurrency models and give a uniform account of concurrency, both in the case of (possibly timed) interleaving and of true concurrency.

The basic idea stems from language and automata theory. Given an automaton, its behaviour can be modelled as the sets of strings over an alphabet A that can be accepted from any of its states. Thus, successful experiments relative to a pair of states (z, z'), are the set of observed (labelled) computations between z and z'; this set of successful experiments is usually referred as 'the language from z to z''. A language is a subset of A^*, and the study of automata can be transferred to the calculus of (regular) languages. This approach – which leads us to obtain computations, not as the result of a complete transition 'from an initial to a terminal state', but as the concatenations of the computations obtained from the transitions from any state to any other state (Betti and Kasangian 1985) – will be referred to here as 'local', in accordance with classical categorical terminology.

When we consider concurrent systems, we can see a concurrent agent as an automaton, where non-determinism derives from the fact that its states can offer a different behaviour at different moments in time. Non-deterministic computations between a pair of states can then no longer be described as a 'set' of strings in a free monoid, because apparently equal steps can lead to different computations. Consequently, between two states we will not have a set of strings but a *labelled structured set of computations*, where the structure describes the possibility of two computations parting from each other while maintaining the same observable steps. We shall assume that the behaviour of an *agent* can be explored by proposing a set of experiments belonging to a given *observation domain*, that is, we will consider the impact of each element of the observation domain on the considered agent. Given an agent X, the result of a single successful experiment on X will be considered as a computation performed by X to become another agent U. Computations are labelled by elements of the observation domain corresponding to successful experiments. The labelling provides 'observational' information, called the *extent* of a computation. But, we also have 'relational' information, called the *agreement*, relative to pairs of computations. For any two pairs of computations u and v, $agreement(u, v)$ specifies how far two computations can be considered indistinguishable via experiments. Extent and agreement induce a tree-like structure on the *computations* of an agent that records all successful experiments performed on it. The *behaviour* of an agent (its 'possible future') is the *full* set of structured computations that it may perform. To provide a local view of systems, it is important to consider the structured set of computations between two agents X and U, that is, to the fragment of the behaviour of X *after which* X behaves like U. The picture on the following page describes, via the 'fat' lines, the structured set of computation from X to U.

In this paper, we shall consider different kinds of observation domains and related structured sets of computations, and will show that they correspond to known algebraic models. Structured sets of computations will be organised as a category of generalised trees built over a meet-semilattice monoid formalizing the observation domain. Theorems allowing us to introduce the usual concurrency operators in the models and relating

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press



different models will then be obtained by first considering ordinary functors (on and between the observation domains) and then lifting them to the categories of structured sets of computations.

Our main aim is to show that the study of labelled structured sets of computations, in analogy with that of automata languages, can be carried in a strict parametrical way both in the interleaving and the true concurrent models, by simply changing the observation domains, that is, the labelling, and not the underlying structure.

As shown in previous papers (Kasangian *et al.* 1987; Kasangian and Labella 1992), and still in analogy with automata theory, we remind the reader that agents belong to a category enriched over a category of structured sets of computations. Any pair of agents can be related by considering the structured set of computations performed by the first to evolve to the second.

We shall start by introducing the observation domains. In Section 2 we will introduce the notion of meet-semilattice monoid and show that *linear* (discrete or continuous) observation domains, and *Mazurkiewicz trace* observation domains, are examples of this monoid. After introducing the observation domain, the crucial definition of observational trees and the morphism between them is given. *Linear* and *trace structured sets of computations* are shown to be the observational trees corresponding to the meet-semilattice monoids mentioned above. They are then used to illustrate some instances of the application of our theory, namely Milner's agents (Milner 1983), Cardelli's continuous time agents (Cardelli 1982) and trees labelled by Mazurkiewicz Traces (Mazurkiewicz 1987) as well as Winskel's event structures (Winskel 1987). A few properties of the category of observational trees are established. In particular, we prove that these categories are equipped with a sum (which naturally models non-determinism) and a (non symmetric) tensor product (which naturally models sequentialization). In the final part of Section 2 we prove a series of theorems that enable us to relate categories of languages and of observational trees, and thus to evaluate our generalization from a categorical point of view.

In Section 3, we start with an account of all the functors that can be defined between categories of observational trees starting from homomorphisms between meet-semilattices. We consider this section as the core of the paper: it shows that the functors relating the different models follow directly from general structural reasoning and are *not* obtained via '*ad hoc*' constructions. In Section 3.2 our theory is re-read in strict categorical terms

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

and used to establish more sophisticated results, which allow a deeper insight into the structural connections between different concurrency models; in particular, it throws light on the interplay between the branching structure of computations and the labelling in different models when modelling non-deterministic behaviours. Some basic definitions of category theory needed to read this section are given in the Appendix.

In Section 4, we present and comment on results obtained for the proposed notion of an operational tree for modelling different aspects of concurrency. In particular, we show how categories of agents can be defined, and how a synchronization operation can be lifted to them starting from a synchronization function. We also explain how our category of observational trees can carry a notion of iteration that is a direct generalization of the Kleene *-operator and which enjoys initiality properties analogous to those of regular languages.

Section 5 is devoted to comparisons with related work that is directed at the same problem.

2. Basic definitions and constructions

In the first part of this section we introduce the basic definitions and the main categorical structures; in the second part we compare these with well-known categories of languages (like those in Betti and Kasangian (1985)).

2.1. *Observation domains*

As mentioned in the introduction, we are interested in viewing structured sets of computations as trees labelled from some observation domain. Hence we first present a structure for modelling *observation domains.*

Definition 2.1. A *meet-semilattice* $(S, \leqslant, \wedge, \bot)$ is a partial order with $\wedge$ the greatest lower bound for any pair of elements, and $\bot$ the bottom element.
A *meet-semilattice* monoid $(S, \bullet, 1, \leqslant, \wedge, \bot)$ is a monoid $(S, \bullet, 1)$ such that the prefix relation $\leqslant$ between its elements (that is, $s \leqslant t$ if and only if there exists $u \in S$ such that $s \bullet u = t$) induces a meet semilattice structure.

The following examples illustrate some of the meet-semilattice monoids we will consider as possible observation domains.

a. Linear observation
A free monoid A^* on an alphabet A is a meet-semilattice $(A^*, \leqslant, \wedge, \epsilon)$, where $s \leqslant t$ if and only if s is a prefix of t and $s \wedge t$ is the maximal common prefix of s and t. The empty word ϵ is the bottom element of the semilattice as well as its identity.

Given an alphabet A of atomic actions, we use $\mathbf{A}$ to denote the corresponding semilattice monoid $(A^*, \bullet, \epsilon, \leqslant, \wedge, \epsilon)$, and we can use it as the observation domain when we are interested in interleaving semantics for *discrete time* models.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

This example can be extended to elementary actions with different duration. Given an alphabet A of atomic actions, a *partial piece-wise constant function* (*pc-function*, for short) f is a function from R^+ (the non-negative real numbers) to A defined on a bounded interval $[0,t) \subseteq R^+$, such that for any $\alpha \in A$, we have $f^{-1}(\alpha)$ is the union of finitely many intervals of the form $[r,s)$, with $r < s$, where both r and s are non-negative real numbers.

Let us denote by CA the set of *pc*-functions. $(CA, \bullet, \Omega, \leqslant, \wedge, \Omega)$ can be viewed as a meet-semilattice where:

(i) $f \leqslant g$ iff if $x \in dom(f)$, then $f(x) = g(x)$,

(ii) $f \wedge g$ is defined as follows:

(a) $dom(f \wedge g) = [0,t)$ where t is such that for all $x < t$ both $f(x)$ and $g(x)$ are defined, $f(x) = g(x)$, and, if $f(t)$ and $g(t)$ are both defined, then $f(t) \neq g(t)$,

(b) $(f \wedge g)(x) = f(x)$ for every $x \in dom(f \wedge g)$.

(iii) The bottom element is the empty function Ω.

CA can also be viewed as a monoid by defining the composition $f \bullet g$ of two *pc*-functions $f : [0,s) \rightarrow A$, $g : [0,t) \rightarrow A$, as follows:

$(f \bullet g)(x) = \textit{if } x < t \textit{ then } f(x) \textit{ else } g(x-t)$.

The identity of $\bullet$ is the empty function Ω. One can easily see that $\leqslant$ is prefix order with respect to this composition.

The corresponding semilattice-monoid **CA** is the observation domain for interleaving semantics in the *continuous time* case. Replacing R^+ by N we would recover the discrete case.

b. Mazurkiewicz traces

A *monoid of traces* on a concurrent alphabet (A, I), where A is a set of events and I is a symmetric and irreflexive (independency) relation, is the monoid $(A^*, \cdot, \epsilon)/\equiv$, where $\equiv$ is the following congruence relation (Mazurkiewicz 1987) :

$s \equiv t$ iff there is a sequence $< s_0, s_1, \ldots, s_n >$ such that:

(i) $s = s_0$,

(ii) $t = s_n$, and

(iii) for every $0 \leqslant i \leqslant n-1$ there are $u_i, v_i \in A^*$ such that $s_i = u_i \alpha \beta v_i, s_{i+1} = u_i \beta \alpha v_i$, and $(\alpha, \beta) \in I$.

A trace $[s]$ is an equivalence class with respect to $\equiv$. $(A^*/\equiv, \leqslant, \wedge, [\epsilon])$ is a semilattice, where, given the traces $[u]$ and $[v]$, we put $[u] \leqslant [v]$ iff for every u_i in $[u]$ there is v_j in $[v]$ such that u_i is a *prefix* of v_j. The meet of the two traces $[u]$ and $[v]$ is the trace $[w]$ such that w is *a* maximal word in the set $\{w_{ij} | w_{ij}$ prefix of u_i and w_{ij} prefix of v_j and $u_i \in [u]$ and $v_j \in [v]\}$. It is easy to see that for any choice of the maximal word w we indeed get the same trace, which is $[w]$. The bottom element is $[\epsilon]$, because the empty string is a prefix of every string. The associated structure is called **TA** and will be the (Mazurkiewicz) trace-observation domain. One could think of a temporal generalization as in the previous case.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

2.2. *Structured sets of computations*

Structured sets of computations will be modelled as trees according to the intuition that they are generalized 'languages' with extra information about agreement.

Definition 2.2. An *observational tree* over a meet-semilattice **S** or **S**-*tree* is a triple $\mathbf{C} = (C, e, a)$, where

— C is a set (the set of paths)
— $e : C \to \mathbf{S}$ is a function called *extent* (the labelling of paths)
— $a : C \times C \to \mathbf{S}$ is a function called *agreement* (the gluing between paths)

such that for every $x, y, z \in C$

— $a(x, y) \leqslant e(x) \wedge e(y)$
— $a(x, x) = e(x)$
— $a(x, y) \wedge a(y, z) \leqslant a(x, z)$
— $a(x, y) = a(y, x)$.

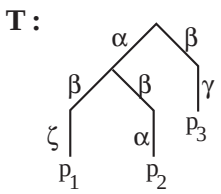
Let us now illustrate observational trees in the cases mentioned above.

a. Linearly labelled structured sets of computations

Let the observation domain be **A** (see **a.** above), then computations between Milner's agents can be viewed as (observational) trees over **A** by considering the structured set of computations that connect pairs of agents (see also Kasangian *et al.* (1987) and Kasangian and Labella (1992)). Here, structured sets of computations will be indicated by the observational tree $\mathbf{T} =< T, e, a >$, where the elements of T are denoted by p_i.

– Every computation p in T is labelled by a word in A^*.
– Given any two computations p_1 and p_2, $a(p_1, p_2)$ is a common prefix of $e(p_1)$ and $e(p_2)$.

As an example, let us take $T = \{p_1, p_2, p_3\}$, $A = \{\alpha, \beta, \gamma, \zeta\}$, $e(p_1) = \alpha\beta\zeta$, $e(p_2) = \alpha\beta\alpha$, $e(p_3) = \beta\gamma$, $a(p_1, p_2) = \alpha$, $a(p_2, p_3) = a(p_1, p_3) = \epsilon$, (and, of course, $a(p_i, p_i) = e(p_i)$ for $i = 1, 2, 3$). They give rise to the structured set of computations $\mathbf{T} =< T, e, a >$ depicted in the figure below, where the elements of T are written in 'leaf positions'.

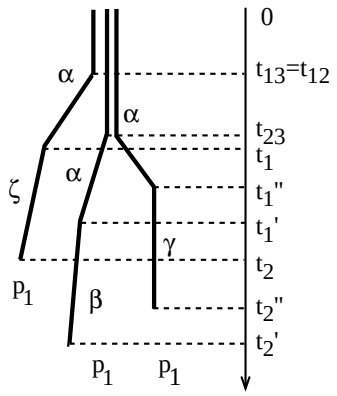


If we consider agents that may perform actions with temporal *duration*, as in Cardelli (1982), we adopt the observation domain **CA** of *pc-functions* (see **a.** above). A structured set of computations corresponding to **CA**, is, for example, a continuous tree with three paths p_1, p_2, p_3 labelled by f_1, f_2 and f_3, respectively,

$f_1^{-1}(\alpha) = [0, t_1)$, $f_1^{-1}(\zeta) = [t_1, t_2)$
$f_2^{-1}(\alpha) = [0, t'_1)$, $f_2^{-1}(\beta) = [t'_1, t'_2)$

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

$f_3^{-1}(\alpha) = [0, t_1'')$, $f_3^{-1}(\gamma) = [t_1'', t_2'')$
$a(p_1, p_2) = \alpha$ on $[0, t_{12})$, $a(p_2, p_3) = \alpha$ on $[0, t_{23})$, $a(p_1, p_3) = \alpha$ on $[0, t_{13})$
can be represented as follows:



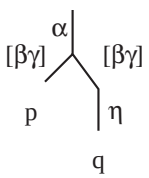
The duration of an action is represented by its projection on the real axis and agreement between paths is indicated by their running in parallel.

b. Mazurkiewicz trace labelled structured sets of computations

We can use **TA** to label our observational trees. Notice that in this case they cannot be depicted as actual trees any longer; bifurcations are not points as in the previous (discrete) case:

$$A = \{\alpha, \beta, \gamma, \eta\}\ I = \{(\beta, \gamma), (\gamma, \beta)\}\ T = \{p, q\}$$
$$e(p) = \alpha\beta\gamma\ e(q) = \alpha\beta\gamma\eta\ a(p, q) = \alpha\beta$$

where by $[\beta\gamma]$ we indicate that the order in $\beta\gamma$ is invertible. This example illustrates the fact that we represent trees without relying on the classical notion of node; it is our notion of agreement that enables us to obtain 'trees' from sets of traces. We cannot see how classical, node-based trees, could be introduced to associate non-deterministic behaviour to Mazurkiewicz traces.



c. Event structures

Let us now consider a different case. Recall that $< E, \leqslant, \# >$ is said to be a (*prime*) *event structure* (Nielsen *et al.* 1981) iff

(i) E is a set of so-called *events*,

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

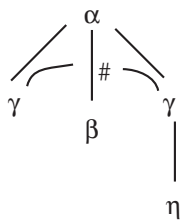
(ii) $\leqslant$ and $\#$ are two binary relations on E such that $\leqslant \cap \# = \varnothing$,
(iii) $\leqslant$ is a partial order such that $\forall e \in E\ \{e' | e' \leqslant e\}$ is finite, and
(iv) $\#$ is symmetric, irreflexive, and hereditary (that is, if $e \# e'$ and $e' \leqslant e''$ then $e \# e''$).

For an event structure we can define a *concurrency* relation co as $(E \times E)/(\leqslant \cup \geqslant \cup \#)$, that is symmetric and irreflexive.

An event structure is said to be *conflict free* if $\# = \varnothing$; a *configuration* in E is a conflict free, downwards closed subset of E.

It is well known, and immediately seen, that a conflict free event structure F can be viewed as a trace over the concurrent alphabet consisting of the labels occurring in it and the concurrency relation defined over it and *vice versa*. One can also show that an event structure can be viewed as an observational tree labelled over this structure and with bifurcation representing conflict. In fact, if L is the language of an event structure E, we have $(L^*, \cdot, \epsilon)/co$ is a trace monoid. Let X be the set of its maximal (with respect to inclusion) configurations: they are conflict free event structures, therefore they are naturally thought of as labelled by traces. Agreement between two of them corresponds to the maximal common subconfiguration.

But trace labelled trees are more expressive then event structures: the picture proposed as an example of trace labelled tree above, can be viewed as the description of the following 'labelled' event structure, where vertical lines denote order and # conflict:



Connections between trace languages and event structures are well known. They have been studied, for instance, in Bednarczyk (1988) and Nielsen *et al.* (1981). In Winskel and Nielsen (1995) it is claimed that labelled event structures and trace languages correspond to different intuitions, in that labelled event structures generalize synchronization trees, while trace languages generalize languages. We will see here that the notion of observational tree, can encompass, in a very natural way, both trace languages and a kind of labelled event structures.

2.3. *The category of observational trees*

Morphisms between trees are functions from paths to paths that: (i) preserve labels, and (ii) have images that are 'glued together' at least as much as in their domain.

Definition 2.3. A *morphism* $f : \mathbf{C}_1 \to \mathbf{C}_2$ between $\mathbf{S}$-trees $\mathbf{C}_1 =< C_1, e_1, a_1 >$ and $\mathbf{C}_2 =< C_2, e_2, a_2 >$ is a function from C_1 to C_2, still denoted by f, such that:

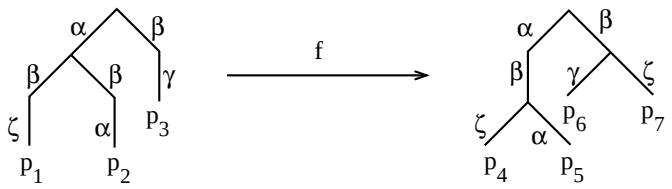
https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

(i) $e_1(x_1) = e_2(f(x_1))$ for any element x_1 in C_1, and
(ii) $a_1(x_1, y_1) \leqslant a_2(f(x_1), f(y_1))$, for any two elements $x_1, y_1 \in C_1$, where $\leqslant$ is the order in **S**.

Example 2.1. Let us consider the tree **T** as above and $\mathbf{T}' =< T', e', a' >$, where

$T' = \{p_4, p_5, p_6, p_7\}$,
$e'(p_4) = \alpha\beta\zeta, e'(p_5) = \alpha\beta\alpha, e'(p_6) = \beta\gamma, e'(p_7) = \beta\zeta$,
$a'(p_4, p_5) = \alpha\beta, a'(p_7, p_6) = \beta, a'(p_4, p_6) = a'(p_5, p_6) = a'(p_7, p_4) = a'(p_7, p_5) = \epsilon$.

The function $f : T \to T'$, such that $f(p_1) = p_4$, $f(p_2) = p_5$, and $f(p_3) = p_6$ is a morphism between $< T, e, a >$ and $< T', e', a' >$.



Definition 2.4. We use **STrees** to mean the category of observational **S**-trees with the corresponding morphisms and composition defined as function composition (hence **ATrees**, **CATrees** and **TATrees** will correspond to the observation domains **A**, **CA** and **TA** respectively).

Facts.

(i) The category **STrees** has coproducts and initial object. We have that the sum of $\mathbf{C}_1 =< C_1, e_1, a_1 >$ and $\mathbf{C}_2 =< C_2, e_2, a_2 >$ is $\mathbf{C} =< C_1 + C_2, e_1 + e_2, a >$, where $+$ denotes the disjoint union (that is, the coproduct in the category **Set**) and the agreement function is defined as follows:

$$\begin{array}{lll} a(x_i, y_i) & = a_i(x_i, y_i) & \text{if } x_i \text{ and } y_i \text{ are in the same } C_i \qquad (i = 1, 2). \\ & = \perp & \text{otherwise.} \end{array}$$

The initial object is the empty tree $\mathbf{0} =< \varnothing, \varnothing, \varnothing >$.

(ii) The category **STrees** has products and terminal object. The product $\mathbf{C} =< C, e, a >$ of the trees $\mathbf{C}_1 =< C_1, e_1, a_1 >$ and $\mathbf{C}_2 =< C_2, e_2, a_2 >$ is defined as follows:
$C = \{< x_1, x_2 > | x_1 \in C_1, x_2 \in C_2, e_1(x_1) = e_2(x_2)\}$,
$e(< x_1, x_2 >) = e_1(x_1) \quad (= e_2(x_2))$ and
$a(< x_1, x_2 >, < y_1, y_2 >) = a_1(x_1, y_1) \wedge a_2(x_2, y_2)$, where $x_i, y_i \in C_i \quad (i = 1, 2)$.

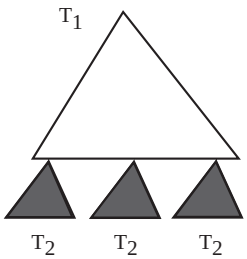
The terminal object is the tree $< S, 1_S, \wedge >$.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

The sum of two objects in **STrees** can be represented by gluing the roots, as in the following picture:

Sum will be used to model non-determinism.

Up to this point we have not exploited the monoidal structure of the different meet-semilattice monoids we had in the examples. If **S** is a meet-semilattice monoid, we can define a *concatenation* operator on **STrees** (Bergstra and Klop 1985). Essentially concatenating two trees, $\mathbf{T_1}$ and $\mathbf{T_2}$, consists of appending a copy of the second one to every leaf of the first one, as in the following picture of $\mathbf{T_1} \otimes \mathbf{T_2}$:



Proposition 2.1. Let **S** be a meet-semilattice monoid. The operation $\bullet : S \times S \to S$ induces a tensor product $\otimes$: **STrees** $\times$ **STrees** $\to$ **STrees**. Thus, the category **STrees** is a monoidal category (Eilenberg and Kelly 1965).

Proof. In **STrees** the concatenation $\otimes$ is defined as follows. For any two observational trees $\mathbf{C}_1 =< C_1, e_1, a_1 >$ and $\mathbf{C}_2 =< C_2, e_2, a_2 >$,

$$\mathbf{C}_1 \otimes \mathbf{C}_2 =< C_1 \times C_2, e_1 \otimes e_2, a_{12} >$$

where for $x_i, y_i \in C_i$ $(i = 1, 2)$,we have

$$\begin{aligned} e_1 \otimes e_2(< x_1, x_2 >) &= e_1(x_1) \bullet e_2(x_2) \\ a_{12}(< x_1, x_2 >, < y_1, y_2 >) &= a_1(x_1, y_1) \bullet a_2(x_2, y_2) \quad \text{if } x_1 = y_1 \\ &= a_1(x_1, y_1) \quad \text{otherwise.} \end{aligned}$$

For any two morphisms $f : \mathbf{C}_1 \to \mathbf{C}_3$ and $g : \mathbf{C}_2 \to \mathbf{C}_4$, where $\mathbf{C}_i =< C_i, e_i, a_i >$ for $i \in \{1, \ldots, 4\}$, $f \otimes g$ is a morphism from $\mathbf{C}_1 \otimes \mathbf{C}_2$ to $\mathbf{C}_3 \otimes \mathbf{C}_4$, that is, from $< C_1 \times C_2, e_1 \otimes e_2, a_{12} >$ to $< C_3 \times C_4, e_3 \otimes e_4, a_{34} >$, such that:

$$\text{for any } x_1 \text{ in } C_1 \text{ and } x_2 \text{ in } C_2,\ (f \otimes g)(x_1, x_2) =< f(x_1), g(x_2) > \text{ in } C_3 \times C_4.$$

It is straightforward to verify that $\otimes$ is an associative functor, and $\mathbf{1} =< \{x\}, e, a >$, where $e(x) = a(x, x) = \epsilon$, is its identity. □

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Remarks.

(i) Our tensor product is *non-commutative*, because it reflects the essential asymmetry of sequential composition.

(ii) **1** corresponds to a single computation with extent ϵ, and **0** has no corresponding computation. One can see that **0** has the disaster, indeed we have $\mathbf{0} \otimes \mathbf{T} = \mathbf{T} \otimes \mathbf{0} = \mathbf{0}$ for any **T** in **STrees**. On the other hand, **1** corresponds to *successful termination.*

(iii) In our model the equation $\mathbf{1} + \mathbf{1} = \mathbf{1}$ does not hold. We could obtain it by adding in Definition 2.2, the extra (skeletality) condition

- $a(x, y) = e(x) = e(y)$ implies $x = y$

We have preferred not to impose this condition, because, with the skeletality condition, **STrees** would fail to be closed in many cases (see Theorem 4.1) and tensor product would not be distributive on the left with respect to sum, as it is natural in order to maintain the analogy with regular languages as far as possible. On the other hand, the presence of an idempotence law $\mathbf{T} + \mathbf{T} = \mathbf{T}$, would be in contrast with the fact that sum is a coproduct.

2.4. *Subcategories of ATrees and languages*

Let us first compare the classical interleaving model for discrete time agents with languages on the same alphabet. Any language (a subset of the free monoid A^*) can certainly be viewed as a tree where

(i.) all agreements are ϵ and

(ii.) all branches have different extent, that is, with no multiplicity on the branches.

Hence languages over a fixed alphabet A constitute a subcategory **Alang** of **ATrees**; morphisms are just language inclusions. In fact, the inclusion functor factors through the subcategory **Atrees** of **ATrees** where trees are *without* multiplicities on the branches, but still with non-trivial agreements. Let us denote by I : **Alang** $\rightarrow$ **Atrees** the inclusion functor.

We also define a forgetful functor V : **Atrees** $\rightarrow$ **Alang**, that associates to each tree the language of the words labelling its branches.

Proposition 2.2. V is right adjoint to I, and hence **Alang** is a coreflective subcategory of **Atrees**. Furthermore, V also has a right adjoint N : **Alang** $\rightarrow$ **Atrees**, and hence **Alang** is also a reflective subcategory of **Atrees**.

N : **Alang** $\rightarrow$ **Atrees** associates with each language the tree with maximal agreements among the branches. The construction is straightforward: given two objects b_1 and b_2 in the discrete **A**-tree **L** corresponding to the language L, their agreement $N(\mathbf{L})(b_1, b_2)$ is just the *maximal* common prefix of the two words. The construction is evidently functorial because inclusions are preserved.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

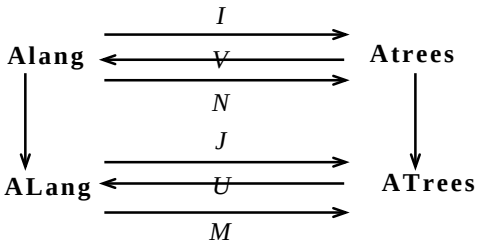
If we drop the requirement of multiplicity on the branches, but keep the one on trivial agreements, we obtain a coreflection between the subcategory of trees with trivial agreements (multilanguages **ALang**) and **ATrees**, with an inclusion functor J : **ALang** → **ATrees** and a forgetful functor U, that simply forgets agreements and keeps multiplicities, hence giving multilanguages. Of course, I and V are restrictions to the subcategory **Atrees** of J and U, respectively.

There is also a functor M : **ALang** → **ATrees**, associating with each multilanguage the tree with maximal agreements among the branches, so N is the restriction of M to **Alang**.

Proposition 2.3. U is right adjoint to J, hence **ALang** is a coreflective subcategory of **ATrees**. Furthermore, U also has a right adjoint M : **ALang** → **ATrees**, hence there is a reflection from **ALang** to **ATrees**.

Remark 2.1. Thus, given a language we have a free and a cofree tree, or a discrete and a codiscrete one and the same happens with multilanguages. The adjunctions $J \dashv U \dashv M$ restrict to $I \dashv V \dashv N$ from **Alang** to **Atrees**.

U is right inverse to both M and J, as V is right inverse to both N and I.



Proposition 2.4. M and U (N and V) are monoidal functors (Eilenberg and Kelly 1965).

Similar results can be proved in the general case of **STrees**, and, therefore, for the obvious generalization of languages to continuous time and for trace languages.

3. Relating different models

3.1. *Changing observation domains*

The aim of our work is to provide a general framework for describing concurrency models. Here, we show that the algebraic structure we rely on enables us to capture naturally different models as instances of our general framework, while permitting a smooth (*uniform*) transition between the different models by means of morphisms between different observation domains. In this way, the intrinsic relations between different models become more evident. Below, we introduce the required morphisms.

Definition 3.1. If **S** and **S′** are meet-semilattices, a *local meet-semilattice homomorphism* f : **S** → **S′** is a monotonic function such that if two elements have a common upper bound, their meet is preserved.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Proposition 3.1. *(Change of base)* A local meet-semilattice homomorphism $\mathbf{S} \to \mathbf{S}'$ induces a functor $F : \mathbf{STrees} \to \mathbf{S}'\mathbf{Trees}$.

Proof. Monotonicity is sufficient to define F in the natural way. Given $\mathbf{X} = (X, e, a)$, $F(\mathbf{X}) = (X, e', a')$, where $e' = fe$ and $a' = fa$. The condition on preservation of meets is needed to satisfy triangular inequality, because in that case $a(x, y)$ and $a(y, z)$ are dominated by $e(y)$ and, therefore, their meet is preserved. Given a morphism h, $F(h)$ is the same function.

Let us start by changing alphabet, that is by considering a *literal* monoid morphism (that is, one induced by a function between the alphabets (Eilenberg 1974)) $l : A^* \to A'^*$ between the corresponding free monoids. l is also a local meet-semilattice homomorphism, so relabelling on trees $L : \mathbf{ATrees} \to \mathbf{A}'\mathbf{Trees}$ is an instance of Proposition 3.1 above.

Proposition 3.2. Though, in general, there is no right adjoint to a functor induced by a local meet-semilattice homomorphism, we have that in the case of an injective relabelling, L has a right adjoint, namely the functor that relabels back paths using l^{-1}, deleting every path whose label is not covered by the function l.

The relabelling functor can also be defined for traces, but in this case we have to require that the function h between alphabets is such that if α and β are independent, then $h(\alpha) \neq h(\beta)$. This requirement is strictly related to the 'absence of autoconcurrency', which is usually imposed on Mazurkiewicz's traces: here it is crucial just to prove that we do indeed have a local meet-semilattice homomorphism and hence the change of base is possible.

Proposition 3.3. If $\mathbf{S}$ and $\mathbf{S}'$ are both meet-semilattices monoids and the local meet-semilattice homomorphism $f : \mathbf{S} \to \mathbf{S}'$ is a monoid homomorphism, the corresponding functor $F : \mathbf{STrees} \to \mathbf{S}'\mathbf{Trees}$ is a monoidal functor (Eilenberg and Kelly 1965).

Therefore the property of being a local meet-semilattice homomorphism is needed to guarantee the tree structure, while preservation of composition is needed to preserve concatenation.

Up to this point we have considered models for concurrency where functions between observation domains are local meet-semilattice homomorphism and monoid homomorphism as well, but, as we shall see later, there are interesting models where only the first property holds.

Let us now use the change of base machinery to investigate the relationships between discrete and continuous time structured sets of computations. As remarked in Section 2, if in **CA** we replace R^+ with N, we obtain the definition of **A**. In other words, *pc*-functions defined on N are equivalent to elements of A^*. Therefore **A** is a sub meet-semilattice monoid of **CA**.

Proposition 3.4. The inclusion homomorphism $c : \mathbf{A} \to \mathbf{CA}$ has a left inverse d that is a local meet-semilattice homomorphism.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Proof. (Sketch) Given a *pc*-function f, we have $d(f) = \alpha_1^{n_1} \ldots \alpha_k^{n_k}$, where α_i are occurrences of values of f in an increasing order and n_i is the maximal natural number in $|[r,s)|$, with $[r,s) = f^{-1}(\alpha_i)$.

Corollary 3.1. Besides the obvious inclusion functor C : **ATrees** → **CATrees** induced by the inclusion i, we have a 'discretization functor' D : **CATrees** → **ATrees** as one of its left inverses (but not an adjoint).

More interesting is the relationship between linear and trace labelled structured sets of computations. Given a concurrent alphabet (A, I), we consider the categories **ATrees** and **TATrees**. There is a canonical monoid homomorphism from A^* to $A^*/\equiv$, which is induced by the congruence $\equiv$ generated by the independence relation I. This homomorphism is still a local meet-semilattice homomorphism **A** → **TA** and induces a functor tr : **ATrees** → **TATrees**.

Thus, for any given **A**-tree $\mathbf{T} =< T, e_T, a_T >$ in **ATrees**, we get the **TA**-tree $tr(\mathbf{T})$, with the same paths. Words that label the paths of the tree are replaced by the corresponding traces (see figure below; where for each path $p \in T$ we use $[[p]]$ to denote the corresponding path in $tr(\mathbf{T})$).

Let us now consider the functor int : **TATrees** → **ATrees**, called the *interleaving* functor. Intuitively, given a tree with paths labelled by traces, the functor int constructs a tree with paths labelled by all strings belonging to those traces. A path labelled by a trace is split by int into a set of paths labelled by strings in the trace (interleaving). The agreement between two paths with labels s_1 and s_2 is the maximal initial common sub-string of s_1 and s_2, which is not longer than the agreement of the corresponding traces. Functor int is formally defined as follows:

— Given the **TA**-tree $\mathbf{Z} =< Z, e_Z, a_Z >$ (that is a tree labelled by traces in $A^*/\equiv$), $int(\mathbf{Z}) =< Z', e_{Z'}, a_{Z'} >$ is the **A**-tree (a tree labelled by words in A^*) such that:
  - $Z' = \{< p, s > | p \in Z, s \in e_Z(p)\}$,
  - $e_{Z'}(< p, s >) = s$,
  - $a_{Z'}(< p_1, s_1 >, < p_2, s_2 >) = (s_1 \wedge s_2)$ restricted to the length of $a_Z(p_1, p_2)$.

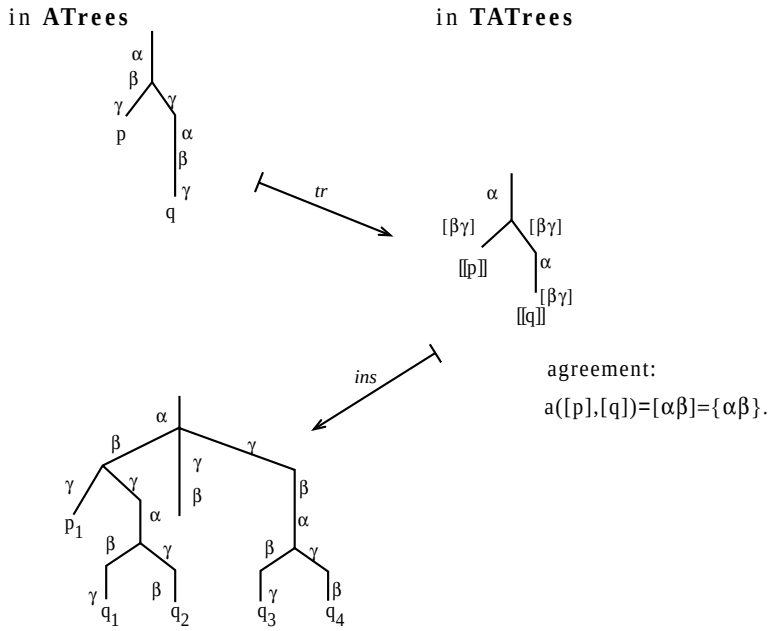
— Given the **TA**-morphism f : $\mathbf{Z_1} \to \mathbf{Z_2}$, $int(f)$: $int(\mathbf{Z_1}) \to int(\mathbf{Z_2})$ is the **A**-morphism defined as follows: $int(f)(< p, s >) =< f(p), s >$ (where f here denotes also the function corresponding to the morphism f).

We have that int is a functor and, for any f, $int(f)$ preserves the extents of paths.

The figure below provides an example of successive applications of tr and int to an **A**-tree: the alphabet is $A = \{\alpha, \beta, \gamma\}$ and the independency relation is $I = \{< \beta, \gamma >, < \gamma, \beta >\}$:
The two leftmost paths in the bottom tree represent the image of the original tree **T** as a subtree of $int(tr(\mathbf{T}))$ via the canonical morphism arising from the adjunction between functors int and tr.

Theorem 3.1. The functor int is right adjoint to tr.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

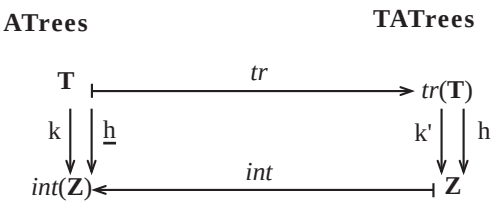


Proof. Let us consider an **A**-tree $\mathbf{T} =< T, e_T, a_T >$ and a **TA**-tree $\mathbf{Z} =< Z, e_Z, a_Z >$.
Let $tr(\mathbf{T})$ be $< T'', e_{T''}, a_{T''} >$ and $int(\mathbf{Z})$ be $< Z', e_{Z'}, a_{Z'} >$.
Recall that: $T'' = T$, and that $[[p]]$ denotes the path in T'' corresponding to path p in T.
Given $k : \mathbf{T} \rightarrow int(\mathbf{Z})$ we define $k' : tr(\mathbf{T}) \rightarrow \mathbf{Z}$ as follows:

$$k'([[p]]) =_{def} \pi_1(k(p)) \text{ for any } p \in T.$$

Conversely, given $h : tr(\mathbf{T}) \rightarrow \mathbf{Z}$ we may define $\underline{h} : \mathbf{T} \rightarrow int(\mathbf{Z})$ as follows:

$$\underline{h}(p) =_{def} < h([[p]]), e_T(p) > \text{ for any } p \in T. \quad \text{Thus, we have}$$



One can easily show that k' is indeed a **TA**-morphism, because
(i) $e_{T''}([[p]]) = e_Z(k'([[p]]))$, and
(ii) $a_{T''}([[p]], [[q]]) \leqslant a_Z(k'([[p]]), k'([[q]]))$,
and h is an **A**-morphism, because:

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

(i) $e_T(p) = e_{Z'}(\underline{h}(p))$, and
(ii)$a_T(p,q) \leqslant a_{Z'}(\underline{h}(p), \underline{h}(q))$.
We have that $\underline{(k')} = k$, and $(\underline{h})' = h$. The naturality conditions are easily checked. □

3.2. *Fibrations and event structures*

The reader acquainted with the notion of enriched category will have certainly noticed that we are in a particular case of enrichment over a *locally posetal 2-category* (Walters 1981):

— a meet-semilattice **S** is a particular locally posetal 2-category (see the Appendix)
— an **S**-tree is a symmetrical category (enriched) over **S** and an **S**-morphism is an **S**-functor.

From this point of view, considering the well-known correspondence between enrichments and fibrations (Betti *et al.* 1983), we could rephrase and extend the result about relabelling in terms of fibrations. Let us also recall that a *split op-fibration* over a category **C** is a functor from **C** to **Cat** (Gray 1966). We know that a category **ATrees** can be associated with every free monoid A^*, considered as a posetal 2-category **A**. Similarly, a category **CATrees** is associated with every **CA** and a category **TATrees** is associated with **TA**.

Let us use **LO** (Linear Observation domains) to denote the category whose objects are posetal 2-categories induced by free monoids and whose arrows are homomorphisms of 2-categories (see the Appendix) induced by *literal* monoid morphisms between the corresponding monoids. Analogously, we can consider **CO** (Continuous Observation domains) and, more interestingly, **TO** (Mazurkiewicz Trace Observation domains), which are the corresponding categories in the other cases defined with the morphisms used above for the change of base. Therefore we have the following theorem.

Theorem 3.2. The construction of trees over a free monoid (semilattice of *pc*-functions), a monoid of traces, respectively, amounts in every case to a split *op*-fibration given by the following functors

$$\mathbf{tree_L} : \mathbf{LO} \rightarrow \mathbf{Cat}$$
$$\mathbf{tree_C} : \mathbf{CO} \rightarrow \mathbf{Cat}$$
$$\mathbf{tree_T} : \mathbf{TO} \rightarrow \mathbf{Cat}$$

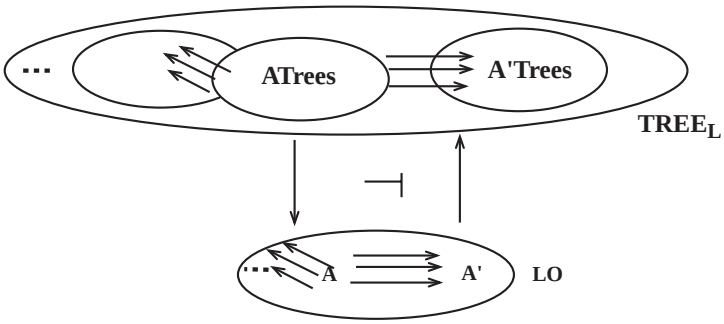
Proof. The correspondence between objects is immediate:
$\mathbf{tree_L}(\mathbf{A}) = \mathbf{ATrees}$, $\mathbf{tree_C}(\mathbf{CA}) = \mathbf{CATrees}$, $\mathbf{tree_T}(\mathbf{TA}) = \mathbf{TATrees}$.
The change of base gives the correspondent of a morphism. The verifications are routine. □

The functors above can be described as categories. For the first case, we will use $\mathbf{TREE_L}$ to denote the category of observational trees over a *variable* free monoid. Objects of $\mathbf{TREE_L}$ can be described as pairs $(\mathbf{X}, \mathbf{A})$, where **A** is the meet semilattice monoid

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

associated with the alphabet **A**, and **X** is an **A**-tree. Morphisms of $\mathbf{TREE_L}$ are pairs $(u, f) : (\mathbf{X}, \mathbf{A}) \to (\mathbf{X}', \mathbf{A}')$, where $f : \mathbf{A} \to \mathbf{A}'$ is a morphism of **LO**, and $u : F(\mathbf{X}) \to \mathbf{X}'$ is a morphism in $\mathbf{A}'$**Trees**. Given $(v, g) : (\mathbf{X}', \mathbf{A}') \to (\mathbf{X}'', \mathbf{A}'')$, with $g : \mathbf{A}' \to \mathbf{A}''$ and $v : G(\mathbf{X}') \to \mathbf{X}''$, the composite is given by $(vG(u), gf)$.

The fibred structure of $\mathbf{TREE_L}$ is reminiscent of a similar one in the context of enriched categorical automata theory (Betti and Kasangian 1985). An analogous description can be given in the trace case, that is, for $\mathbf{TREE_T}$. Theorem 3.2 states that by changing alphabet we obtain a very smooth transformation on the corresponding trees. Below we give a schematic picture in the case of **LO**; the other cases are similar.



We can use the categories obtained above as fibrations to investigate more deeply the comparison between trace labelled trees and the event structures considered in Section 2, seen as a tree labelled by traces. We shall prove that a 'suitable' category of event structures is equivalent to a subcategory of the fibration $\mathbf{TREE_T}$.

We will now define the category **ES** of event structures. Objects are event structures, as defined in Section 2, morphisms are defined as follows.

Definition 3.2. Given two event structures E and F, morphisms of event structures are functions $h : E \to F$ such that

(i) If C is a maximal configuration (with respect to inclusion) of E, then $h(C)$ is a maximal configuration of F.
(ii) h reflects the relations '$<$' and '$=$' locally, that is, if $e, f \in C$, then

— If $h(e) < h(f)$, then $e < f$
— If $h(e) = h(f)$, then $e = f$

Notice that our definition of morphisms is slightly more demanding than Winskel's (Winskel 1987). In particular, we require morphisms to be total functions. More informally, Condition (i) is needed to preserve the extent of a path in a tree, while (ii) is needed to have 'good morphisms' (the ones of $\mathbf{TREE_T}$) between the correspondent trace monoids. We call it *local* because it reflects relations when restricted to a maximal configuration. As a consequence of (i) and (ii), h preserves concurrency and reflects conflicts.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Definition 3.3. Let us define **TSC** ('good' Trace labelled Structured sets of Computations) as the full subcategory of $\mathbf{TREE_T}$ made out of *superskeletal* trees, that is, those for which it never happens that $a(x, y) = e(x)$.

By considering **TSC** instead of $\mathbf{TREE_T}$, we exclude some (in this case) pathological structures, namely those where a path is completely glued to another path. This would mean that a maximal configuration would be completely contained in another maximal configuration.

In this context, we can establish a result similar to that reported in Winskel and Nielsen (1995, Section 8.3.2), where an adjunction is defined between a category of event structures and a category of trace languages. We will show how to define an adjunction in our case, but we will also point out that the branching structure of trace-labelled trees does not play an essential role; agreement between paths in a tree obtained from an event structure is always obtained as the maximal one compatible with the labelling. This means that, as pointed out in Winskel and Nielsen (1995), in contrast with labelled event structures, trace languages and event structures do not properly take non-determinism into account. Nevertheless, it is challanging to investigate how far trace-labelled trees can go toward modelling non-determinism. The last theorem of this section will be the answer to this question.

We now define a functor $\Theta : \mathbf{ES} \to \mathbf{TSC}$ that describes an event structure as a tree of traces. Given an event structure $(E, \leqslant, \#)$, consider the trace monoid **TE**, where E is viewed as an alphabet and E^* is quotiented out by the congruence generated by the concurrency relation *co*. It is well known (see, for example, Winskel and Nielsen (1995)) that configurations correspond to objects of **TE**.

— On the objects, Θ is defined by $\Theta(E) = (\mathbf{X_E}, \mathbf{TE})$, where $\mathbf{X_E} = (\{C_i\}, e, a)$ is the **TE**-tree whose paths are *maximal* configurations of E. The extent $e(C_i)$ is just the trace $[f]_i$, where f is any representative of C_i viewed as a trace. The agreement $a(C_i, C_j) = [g]_{ij}$, where g is a representative of $C_i \cap C_j$.
— As for the morphisms, observe first that an arrow $h : E \to F$ in **ES** induces a literal morphism $\mathbf{TE} \to \mathbf{TF}$ (still called h), that is a local meet-semilattice morphism in **TO**, because if α and β are independent, then $h(\alpha) \neq h(\beta)$, is injective when restricted to a configuration and preserves the independency relation because it reflects order and conflict.

To obtain an arrow in **TSC** from $\mathbf{X_E}$ to $\mathbf{X_F}$, we still need a **TF**-morphism from $H(\mathbf{X_E})$ to $\mathbf{X_F}$ in **TFTrees**, where $H(\mathbf{X_E})$ has the same objects as $\mathbf{X_E}$, $e_{H(\mathbf{X_E})}(x) = h(e_{\mathbf{X_E}}(x))$ and $a_{H(\mathbf{X_E})}(x_1, x_2) = h(a_{\mathbf{X_E}}(x_1, x_2))$. Again, h induces a function from $H(\mathbf{X_E})$ to $\mathbf{X_F}$, since it maps maximal configurations into maximal configurations, and it is in fact a **TF**-morphism because h reflects conflicts, and hence preserves agreement.

To define a functor $\Gamma : \mathbf{TSC} \to \mathbf{ES}$:

— On the objects, we will associate with every $(\mathbf{X}, \mathbf{TA})$ in **TSC** an event structure $(E, \leqslant_E, \#_E)$, where E is the set $\{\alpha_{ki}\}_{i \in J}$, of the letters α occurring in the extents $e(x_i)$

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

of **X**-paths, together with a double indexing according both to the path i and to the occurrence order k, and quotiented out by the equivalence relation ρ defined as follows: for all i and j in J, $\alpha_{ki}\ \rho\ \alpha_{kj}$ iff α_k occurs in $\alpha(x_i, x_j)$.

The conflict relation $\#_E$ is defined by: for all i and j in J, $\alpha_{ki} \#_E \alpha_{lj}$ iff $i \neq j$ and α_k and α_l do not occur in $\alpha(x_i, x_j)$. The following relations are defined over occurrences and then induced over equivalence classes.

The order relation $\leqslant_E$ is induced by $\alpha_{ki} \leqslant_E \alpha_{lj}$ iff $i = j$ and $k \leqslant l$.

— A **TSC** morphism $(u, f) : (\mathbf{X}, \mathbf{TA}) \to (\mathbf{X}', \mathbf{TA}')$ induces a function between the corresponding event structures defined as $\Gamma(u, f)(\alpha_{ki}) = f(\alpha)_{ku(i)}$, which satisfies the requirements of Definition 3.2:

- It is well defined because of the non-autoconcurrency condition.
- It preserves maximal configurations because u preserves extents.
- It locally reflects '$<$' and '$=$' because f is a morphism of traces.

With all this, it is straightforward to prove the following theorem.

Theorem 3.3. The functor Γ : **TSC** $\to$ **ES** is right adjoint to Θ. Furthermore, **ES** is isomorphic to a coreflective subcategory of **TSC**.

Theorem 3.3 establishes that an event structure corresponds to a particular choice of conflicts in the world of its trace alphabet, because the agreement is always the maximal one: the functor Θ selects this one. To consider actions in traces as events, Γ distinguishes between different occurrences of the same action and defines ordering and conflict according to the tree structure. Hence we have that trace-labelled trees are considerably more expressive than (non-labelled) event structures with respect to non-determinism (see Example b. of Section 2.2). On the other hand, they are not as expressive as labelled event structures (general 2-categories woud be required to achieve this).

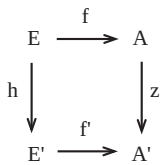
Actually trace-labelled trees lie between labelled and non-labelled event structures. We will show that **TSC** is equivalent to the category of strictly labelled event structures, that is, event structures where the labelling function strictly preserves concurrency in the sense of Definition 3.4 below, where Condition (i) is the non-autoconcurrency condition of Mazurkiewicz traces and Condition (ii) is a sort of preservation of concurrency.

Definition 3.4. A *strictly labelled event structure* $< E, \leqslant, \#, A, f >$ is an event structure $< E, \leqslant, \# >$ with an aphabet A and a labelling function $f : E \to A$ such that, if $e\ co\ e'$

(i) $f(e) \neq f(e')$

(ii) $f(\underline{e}) = f(e)$ and $f(\underline{e'}) = f(e')$ implies that $\underline{e}$ and $\underline{e'}$ are not consecutive in $<$.

A *morphism* between two *strictly labelled event structures* $< E, \leqslant, \#, A, f >$ and $< E', \leqslant', \#', A', f' >$ is an event structure morphism $h : E \to E'$ and a function $z : A \to A'$ such that the following diagram commutes:



https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

sLES is the category of strictly labelled event structures with their morphisms.

Theorem 3.4. There is an equivalence between **TSC** and **sLES**.

Proof. We can essentially re-use the proof of Theorem 3.3. Given $< E, \leqslant, \#, A, f >$, to define Θ' from **sLES** to **TSC**:

— Let us consider **TA** as the trace monoid A^* quotiented via the independence relation generated by: aIa' if there exist e and e', such that $e\ co\ e'$ and $a = f(e)$, $a' = f(e')$. This relation is well defined because, due to the strictness Condition (i) for f, we have that aIa will never happen.
Then the trace labelled tree corresponding to $< E, \leqslant, \#, A, f >$, **E**, has maximal configurations as paths, each of them labelled via the trace of a linearization of its labelled events; this trace is well defined because, by strictness Condition (ii), two consecutive events will never have independent labels.
The agreement is given by the labelling of their intersection.
A morphism between strictly labelled event structures induces a morphism between the corresponding trees. In fact, maximal configurations are sent into maximal configurations, labelling is transformed according to z, which induces a trace language morphism, because it commutes the diagram above and h preserves co and reflects chains. The agreement in the image is not smaller than the image of the original agreement because h reflects $\#$.

— In the opposite direction, given a trace labelled tree $(\mathbf{X}, \mathbf{TA})$, to define $\Gamma' : \mathbf{TSC} \to \mathbf{sLES}$, we consider the set of all occurrences of labels on its paths as events; as in Theorem 3.3, A is the set of atomic labels with the obvious labelling function f from events to labels; f is indeed a strict labelling. The rest of the proof goes as in Theorem 3.3, but in this case we have that $\Gamma'\Theta'(< E, \leqslant, \#, A, f >) \cong < E, \leqslant, \#, A, f >$ and $\Theta'\Gamma'(\mathbf{X}, \mathbf{TA}) \cong (\mathbf{X}, \mathbf{TA})$. □

4. Further results on the model

So far we have described categories of observational trees extensively. It is possible to use them in many ways for modelling different aspects of concurrency. Let us sketch some applications and results (some of these have appeared before (Kasangian *et al.* 1987; Kasangian and Labella 1992; Corradini *et al.* 1995)).

4.1. *Modelling the world of agents*

As pointed out in the Introduction, and in analogy with automata theory, we can use our category of observational trees as a calculus and can relate pairs of agents (or states) by means of the structured set of computations transforming one into the other. The

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

composition of structured sets of computations is obtained by means of concatenation. We will exploit here the fact that, due to the monoidal structure of **S**, (**STrees**, $\otimes$, **1**) is a monoidal category, therefore a category **Ag** of agents can be thought of as an **STrees**-category in the sense of *enrichment* over a monoidal category, in the same way as an automaton on an alphabet forms a category enriched over the corresponding languages. **STrees**-functors transform categories of agents.

It is possible to show that, if the (left) cancellation property holds for **S**, the monoidal category **STrees** is left-closed, that is, for every object **T**, the concatenation functor $- \otimes \mathbf{T}$ has a right adjoint, to be considered as an internal hom-object functor; as a consequence **STrees** is an **STrees**-category in a canonical sense and we have a canonical class of models for agents on **S**. In simple terms, we have that an agent can be identified with its behaviour.

Definition 4.1. A meet-semilattice monoid $(S, \bullet, 1, \leqslant, \wedge, \perp)$ enjoys the *(left) cancellation property* if for every a, b and c in S: $a \bullet b = a \bullet c$ implies $b = c$

Theorem 4.1. If **S** enjoys the (left) cancellation property, then the monoidal category (**STrees**, $\otimes$, **1**) is left closed and, therefore, it is an **STrees**-category in a canonical way.

Proof. Let us show how the assertion can be proved. The hom-object between any two trees $\mathbf{T_1} =< T_1, e_1, a_1 >$ and $\mathbf{T_2} =< T_2, e_2, a_2 >$ consists of the part of $\mathbf{T_2}$ including all paths from the root of $\mathbf{T_2}$ to a homomorphic copy of $\mathbf{T_1}$. Thus, the hom-object is the structured set of computations leading from the agent $\mathbf{T_2}$ to a homomorphic copy of the agent $\mathbf{T_1}$ (see figure below).
Explicitly, the tree $\mathbf{STrees}(\mathbf{T_1}, \mathbf{T_2}) =< T, e, a >$, where:

— $T = \{\pi_{sC} : \mathbf{T_1} \to \mathbf{C} | \pi_{sC}$ is a morphism, $s \in S$ and $\mathbf{C}$ is a tree$\}$, where:
  - C is a subset of paths in $\mathbf{T_2}$ satisfying the following two conditions:
    (i) for each path $p \in C$ $s \leqslant e_2(p)$,
    (ii)for each pair of paths $p_1, p_2 \in C$ $s \leqslant a_2(p_1, p_2)$, and
  - $e_C(p)$ and $a_C(p_1, p_2)$ are defined via:
  - $e_2(p) = s \cdot e_C(p)$ and
  - $a_2(p_1, p_2) = s \cdot a_C(p_1, p_2)$,

— $e(\pi_{sC}) = s$, and for any pair of isomorphisms π_1, π_2 in T we have:

— $a(\pi_1, \pi_2) = a(p_1, p_2)$, where p_i is a path of $\mathbf{T_2}$ in the image of π_i $(i = 1, 2)$. □

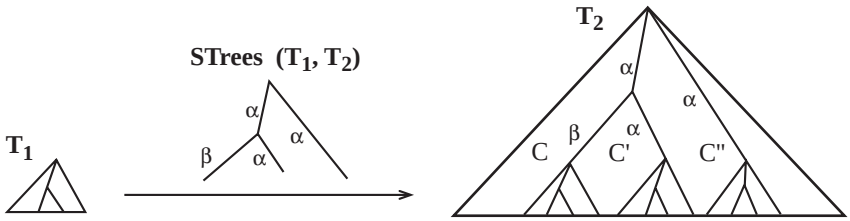
In the figure below we show three isomorphisms corresponding to $s = \alpha\beta$, $\alpha\alpha$, α.

4.2. *Defining synchronization*

It is also possible to define a notion of *synchronization* on **STrees**, that is, between structured sets of computations, as a monoidal functor $* : \mathbf{STrees} \times \mathbf{STrees} \to \mathbf{STrees}$ and, under suitable conditions, to lift the synchronization to computing agents.

As an example, consider *CCS* agents and their unfoldings (Milner 1980, p. 12); these can be viewed as a category of agents in our sense when the observation domain **S** is

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press



the 2-category **A**, and the monoidal functor **ATrees** $\times$ **ATrees** $\rightarrow$ **ATrees** is obtained via a synchronization table.

Proposition 4.1. If, for every object w in **S**, $\{v | v \leqslant w\}$ is totally ordered with respect to $\leqslant$, a partial monoid morphism $* : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}$ (which is defined on the neutral element and is such that whenever $v * w$ is defined and $v' \leqslant v$, then there is $w' \leqslant w$ such that $v' * w' \leqslant v * w$ is defined also) extends to a monoidal functor $* :$ **STrees** $\times$ **STrees** $\rightarrow$ **STrees**.

Proof. (Sketch) The functor $*$ is defined on the objects as follows:

Given the trees $\mathbf{T}_i = \{< T_i, e_i, a_i > \text{ for } i = 1, 2\}$, $\mathbf{T_1} * \mathbf{T_2}$ is the tree $\mathbf{T} =< T, e, a >$ such that:

— $T = \{< p_1, p_2 > | (e(p_1) * e(p_2))$ is defined with $p_i \in T_i$ for $i = 1, 2\}$,

— $e(< p_1, p_2 >) = (e(p_1) * e(p_2))$, and

— $a(< p_1, p_2 >, < q_1, q_2 >)$ is the maximal common prefix w of $e(< p_1, p_2 >)$ and $e(< q_1, q_2 >)$ such that $w = u * v$ where $u \leqslant a(p_1, q_1)$ and $v \leqslant a(p_2, q_2)$.

Functor $*$ is defined on the morphisms as follows:

Given the morphisms $f : \mathbf{T_1} \rightarrow \mathbf{T}_3$ and $g : \mathbf{T_2} \rightarrow \mathbf{T}_4$, where $\mathbf{T}_i =< T_i, e_i, a_i >$ for $i = 1, \ldots, 4$, and p_1 in $\mathbf{T_1}$, and p_2 in $\mathbf{T_2}$ $(f * g) < p_1, p_2 >=< f(p_1), g(p_2) >$.

One can easily check that $*$ is a monoidal functor with respect to the tensor product $\otimes$ (Eilenberg and Kelly 1965). Indeed, **1** is isomorphic to $\mathbf{1} * \mathbf{1}$, and there exists a natural transformation such that for any 4-tuple of trees $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T}_3$ and $\mathbf{T_4}$, the corresponding component is an **STrees**-functor from $(\mathbf{T_1} * \mathbf{T_2}) \otimes (\mathbf{T_3} * \mathbf{T_4}) \rightarrow (\mathbf{T_1} \otimes \mathbf{T_3}) * (\mathbf{T_2} \otimes \mathbf{T_4})$ (Interchange Law). □

Corollary 4.1. When considering **ATrees** (**CATrees**) a *partial* (synchronization) function $* : A \times A \rightarrow A$ extends to a monoidal functor

$$* : \mathbf{ATrees} \times \mathbf{ATrees} \rightarrow \mathbf{ATrees} \ (\mathbf{CATrees} \times \mathbf{CATrees} \rightarrow \mathbf{CATrees}).$$

As a consequence of Proposition 4.1, we know that, using $*$, we can synchronize local behaviours, but it can be proved (Kasangian *et al.* 1987; Kasangian and Labella 1992; Kasangian and Labella 1992a) that in the case of the corollary, there also exists a functor σ that enables us to define the agent that is the synchronization of the two given agents, that is, we can synchronize global behaviours. This last result can be formulated in a general way.

Definition 4.2. (Eilenberg and Kelly 1965) Given a monoidal category $(\mathbf{M}, \otimes, I)$, a

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

monoidal functor $F : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$ and an $\mathbf{M}$-category $\mathbf{C}$, we can always construct the $\mathbf{M}$-category $\mathbf{C} \times_F \mathbf{C}$ (called the *effect* of the monoidal functor): objects are pairs of objects (A, B) in $\mathbf{C} \times \mathbf{C}$ and $\mathbf{C} \times_F \mathbf{C}[(A, B), (A', B')] = F(\mathbf{C}[A, A'], \mathbf{C}[B, B'])$.

Proposition 4.2. Let **STrees** be an **STrees**-category in a canonical sense. Assuming the hypotheses of Proposition 4.1, there exists an **STrees**-functor σ : **STrees** $\times_*$ **STrees** $\to$ **STrees** lifting the monoidal functor $*$: **STrees** $\times$ **STrees** $\to$ **STrees**.

Proof. As before, we simply show how σ can be defined. Given a monoidal synchronization functor $*$ and two trees $\mathbf{T_1} =< \mathbf{T_1}, e_1, a_1 >$ and $\mathbf{T_2} =< \mathbf{T_2}, e_2, a_2 >$, we define an **STrees**-functor σ on objects as follows: $\mathbf{T_1} \sigma \mathbf{T_2} =< T, e, a >$, where:

— $T = T_1 \times T_2$,

— $e(< p_1, p_2 >)$ is the maximal $w = w_1 * w_2$, where w_i is a prefix of $e_i(p_i)$ $(i = 1, 2)$ in **S**, and

— $a(< p_1, p_2 >, < q_1, q_2 >)$ is the maximal common prefix w of $e(< p_1, p_2 >)$ and $e(< q_1, q_2 >)$ such that $w = u * v$ where $u \leqslant a(p_1, q_1)$ and $v \leqslant a(p_2, q_2)$.

The definition of σ on the hom-objects is given by the synchronization functor $*$. □

Intuitively speaking, for example in the case of Milner's agents, the synchronization of two trees using σ is obtained by first synchronizing any two paths, one for each tree as long as it is possible starting from the root using the given synchronization algebra $*$, and then by computing the agreement between the resulting paths using $*$, as defined above.

A different kind of synchronization, related to a notion of non global clock, can also be considered on the model (Kasangian *et al.* 1994).

4.3. *Trees and regular languages*

Let us now restrict to the case of **ATrees**. We can try to exploit the fact that our trees originate from the analogy with language theory and compare them with regular languages as a semantic for regular expressions. To this end, we need to introduce a notion of iteration on **ATrees**.

Definition 4.3. Given a tree $t = (X, e, a)$, we define $t^\infty = (X^\infty, e^\infty, a^\infty)$ as follows:

— $X^\infty = \{< x_1, x_2, \ldots, x_n > | \; n \in N \text{ and } x_i \in X\}$

— $e^\infty(< x_1, x_2, \ldots, x_n >) = e(x_1)e(x_2)\ldots e(x_n)$

— $a^\infty(< x_1, x_2, \ldots, x_n >, < y_1, y_2, \ldots, y_m >) = e(x_1)e(x_2)\ldots e(x_k)a(x_{k+1}, y_{k+1})$ where $k + 1$ is the first index such that $x_{k+1} \neq y_{k+1}$

An approximant $t^j = (X^j, e^j, a^j)$ of t^∞ is defined by:

— $X^j = \{< x_1, x_2, \ldots, x_n > | x_i \in X, \; 0 \leqslant n \leqslant j\}$

— $e^j < x_1, x_2, \ldots, x_n >) = e(x_1)e(x_2)\ldots e(x_n)$

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

— $a^j(< x_1, x_2, \ldots, x_n >, < y_1, y_2, \ldots, y_m >) = e(x_1)e(x_2)\ldots e(x_k)a(x_{k+1}, y_{k+1})$
where $k+1$ is the first index such that $x_{k+1} \neq y_{k+1}$.

Fact. Trees of the form t^∞ are colimits of chains of their approximants as follows:

$$t^0 \to t^1 \to t^2 \to t^3 \to \ldots \to t^{j-1} \to t^j \to t^{j+1} \to \ldots$$

where morphisms are the obvious inclusions (regular monos).

Let us now consider trees as objects of **ATrees** up to isomorphisms.

Definition 4.4. A tree is a *regular tree* on an alphabet A if it is defined as the image of a regular expression P built from $0, 1, A, +, \bullet, ()^*$ with the *non-empty word property* through the following interpretation function i:

$$\begin{aligned} i(0) &= \mathbf{0} \\ i(1) &= \mathbf{1} \\ i(a) &= (\{*\}, a, a) \qquad \text{for every a in } A \\ i(P + P') &= i(P) + i(P') \\ i(P \bullet P') &= i(P) \otimes i(P') \\ i(P^*) &= i(P)^\infty \end{aligned}$$

Let T^∞ be the set of regular trees (up to isos).

Proposition 4.3. If we define in T^∞ a partial order $\leqslant$ as the existence of a regular monomorphism in **ATrees** from the left-hand to the right-hand side, t^∞ is the least fixed point for the endofunctor $F_t : T^\infty \to T^\infty$ defined as follows: $F_t(-) = t \otimes (-) + \mathbf{1}$.

Proof. F_t satisfies the cocompleteness condition required in order to have a minimal fixed point (Smith and Plotkin 1982), that is, $F_t(colim < F_t^n(\mathbf{0}), F_t^n(\mathbf{0}_{Ft(\mathbf{0})}) >) \cong colim\ F_t(< F_t^n(\mathbf{0}), F_t^n(\mathbf{0}_{Ft(\mathbf{0})}) >)$. In fact $t^\infty \cong F_t(t^\infty)$ is such an object and, therefore, the least fixed point. □

Looking for properties enjoyed by $(T^\infty, \leqslant, \otimes, \mathbf{0}, \mathbf{1}, (-)^\infty)$, one finds that a list very similar to the set of axioms of Kozen for Kleene algebras (Kozen 1991) is sound for this interpretation; namely

$$X + Y = Y + X$$

$$(X + Y) + Z = X + (Y + Z)$$

$$X + 0 = X$$

$$(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$$

$$X \bullet 0 = 0 = 0 \bullet X$$

$$X \bullet 1 = X = 1 \bullet X$$

$$(X + Y) \bullet Z = (X \bullet Z) + (Y \bullet Z)$$

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

$$X + Z = Y + Z \Rightarrow X = Y$$

$$1 + X \bullet X^* = X^*$$

$$Z + X \bullet Y \leqslant Y \Rightarrow X^* Z \leqslant Y$$

In the first part of this list, one may notice the lack of the idempotence law for sum and the left distributivity law of concatenation with respect to sum: the loss of the latter is directly connected with non-determinism, while the former is missing and replaced by the cancellation law for sum, because we are dealing with a coproduct. For this first part of the list, our finite trees are an initial algebra.

The final part of the list expresses the fact that iteration is characterizable as a minimal fixed point.

Trees or direct acyclic graphs have often been used as models of languages for concurrency similar to ours, more specifically as models for process algebras like *CCS* (Milner 1989), *CSP* (Hoare 1989) and *ACP* (Bergstra and Klop 1989). However, to the best of our knowledge, none of the proposed structures turns out to be an initial model for the corresponding axiom systems; all of them need a factorization through a suitable notion of behavioural equivalence that, very often, is based on the notion of bisimulation (Park 1981), and in many cases they do not consider all the usual operators of Kleene algebras (see also Rutten (1990), Benson and Tiuryn (1989), Aceto and Hennessy (1992) and Bergstra *et al.* (1987)).

In our case, since sum is obtained as a disjoint union, isomorphism does correspond to an equivalence that takes into account the number of times a transition is possible, and a state is reachable (De Nicola and Labella 1998). This equivalence has been extensively studied in Corradini *et al.* (1995) and called resource bisimulation (it was called counting bisimulation in D'Agostino (1998)).

5. Comparisons and conclusions

If one wants to look for an ancestor for our way of considering trees as sets of paths, one can think of the notion of category of paths on a graph introduced in Goguen (1974). Although we were not aware of that work when we introduced observational trees[†], we find it is in the same spirit as ours.

Our work is also strictly related to Winskel's in more than one respect; we can mention three main topics treated in terms of category theory: synchronization trees, event structures and functorial comparisons between models.

Winskel (Winskel 1987) constructs the categorical counterpart of different models by defining in each case morphisms that make it possible to place suitable functors between them and to look for adjunctions. In this way many categories are defined and related. Our approach is different in two respects:

[†] We would like to thank Rocco De Nicola for having pointed out to us this paper during the fruitful discussions about the present work.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

1. We have a single category of observational trees (whose morphisms are dictated by the enriched framework) that is instantiated to encompass different models by changing the base category. Hence, all comparisons are obtained by lifting simple transformations between the labelling domains. This way of using category theory in a prescriptive way, allows us to throw light on the intrinsic relations between different models of concurrency.
2. Winskel's constructions aim at describing the full behaviours of concurrent agents, while our trees, like languages for automata, are local, in the sense that they describe behaviours between concurrent agents, or, equivalently, between the states of an agent. This basic intuition allows us to use observation domains as an internal logic in the sense of Lawvere (Lawvere 1973). With this fact in mind, one can easily see that the definitions of morphisms in Winskel's work are derivable from our general definition, if one 'adds the conditions due to locality'.

As an example, let us consider the definition of morphism between prime event structures (Winskel 1987):

Definition 3.2′. Given event structures E and F, morphisms of event structures are partial functions $h : E \to F$ such that:

(i) If $h(e)$ is defined, then $(h(e)] \sqsubseteq h(e]$.
(ii) If C is a configuration of E, then $h(C)$ is a configuration of F.
(iii) h is injective if restricted to a configuration.

It is immediate to see that if we require that h in Definition 3.2′ be total and preserve maximal configurations (conditions connected with the consideration of local behaviours), the above definition collapses to our Definition 3.2.

Similar considerations can be made about morphisms between synchronization trees (Winskel 1984). The change of perspective, puts us in a situation where we can claim that our Definition 3.2 of morphism (and the others) is not only suitable, but is induced by the algebraic framework.

A different means of providing a general account of classes of concurrency models is given in Degano *et al.* (1993). Experiments over transition systems are structured as trees whose nodes are labelled by the observations of the corresponding computations. The trees are compared via bisimulations and it is shown that various observational equivalences proposed in the literature can be recast in a general experimental setting. They also study the impact of the chosen observations on the different equivalences.

The nodes are labelled by structures similar to our observation domains; hence the result presented there about the different kind of observation function can be recast in our framework.

They exploit computations, concatenation and prefix ordering over the observation domain are exploited to classify observation functions as follows:

If $o(-)$ is an observation function on computations, then

— $o(-)$ is *incremental* if $o(c) = o(c')$ and $o(d) = o(d')$ implies $o(cd) = o(c'd')$

— $o(-)$ is *cancellative* if $o(c) = o(c')$ and $o(cd) = o(c'd')$ implies $o(d) = o(d')$.

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

In our terminology: the first condition means that their observation functions do not necessarily preserve the composition of computations, therefore their free concatenation does not correspond to the structure of the computations, while it respects their order. To encompass this kind of observation, our model must reasonably give up the monoidal structure of the category though mantaining the order structure and, therefore, the construction of **STrees**. On the other hand, the monoidal structure is not sufficient to guarantee the (left) cancellation property (although all the domains of observation we have considered as examples in the paper are cancellative, a counterexample can be easily provided), hence our trees can also be used to model compositional non-cancellative observations. In that case **STrees** will lose the closure properties shown in Section 4, and their consequences.

In our view, the present work throws light, once again, but formally, on the central role of automata in theoretical computer science. By mimicking the constructions of one of the categorical presentations of automata that relies on an observational account of computation, while taking into account non-determinism, we have shown that many of the mathematical models of concurrency can be captured. There is one notable exception with respect to the models that we can capture with our approach, namely those models based on observations yielding pomsets (partially ordered multisets of actions). The remarks at the end of Section 3 suggest generalizations in this direction. It is well known that pomsets are associated with an observation function that is neither incremental nor cancellative, hence an observation domain for pomsets will only have the order structure. Nonetheless, they will no longer be a meet-semilattice because, due to the multiplicity of elements, inclusions must be specified and we need a more general base 2-category. This will be the subject of further investigation.

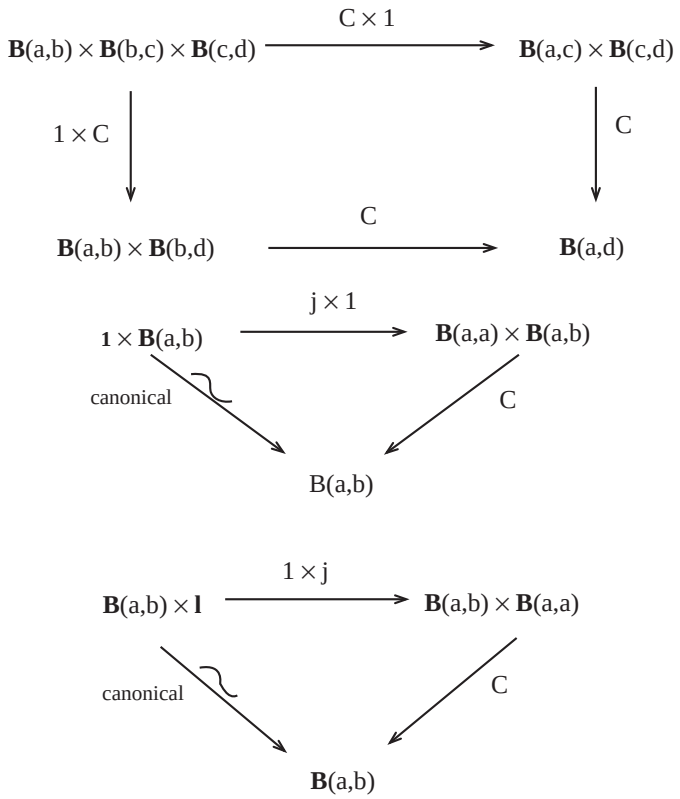
Appendix A.

We will assume that the reader is acquainted with the notions of *tensor product, monoidal category, monoidal functor* and *enrichment over a monoidal category*: if you require details, refer to Eilenberg and Kelly (1965) and Kelly (1982). Given the centrality of the notion of enrichment (or 'category based on') for our treatement in the case of locally posetal 2-categories, we give the following definitions for the sake of completeness.

Definition A.1. (Bénabou 1967) A 2-*category* **B** consists of a class (possibly a set) of objects $Ob(\mathbf{B})$, together with, for each a, b in $Ob(\mathbf{B})$, *a category* $\mathbf{B}(a, b)$, for each triple of objects a, b, c in a $Ob(\mathbf{B})$, a functor (called the *composition* functor) $C : \mathbf{B}(a, b) \times \mathbf{B}(b, c) \to \mathbf{B}(a, c)$ and, for each a in $Ob(\mathbf{B})$, a functor $j : 1 \to \mathbf{B}(a, a)$, where **1** is the trivial one object category, such that the following diagrams commute:

For any a, b in $Ob(\mathbf{B})$ the objects of $\mathbf{B}(a, b)$ are called morphisms of **B** (or 1-cells of **B**), and they will be collectively denoted by $morph(\mathbf{B})$. Analogously, for any a, b in $Ob(\mathbf{B})$ the arrows of $\mathbf{B}(a, b)$ are called 2 -cells of **B**. Composition in any $\mathbf{B}(a, b)$ is called *vertical composition*,

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press



while the composition defined by the functor C is called *horizontal composition*. If f and g are objects of the same category $\mathbf{B}(a, b)$, that is 1-cells with common domain a and common codomain b, then a 2-cell η from f to g may be written as $\eta : f \Rightarrow g$

Notice that functoriality of C yields that $(\alpha\alpha').(\beta\beta') = (\alpha.\beta)(\alpha'.\alpha')$ for 1-cells α and α' (respectively, β and β') with the same domain and codomain and such that dom β =cod α. This is a remarkable interchange law between vertical and horizontal composition, not to be confused with the one betwen the global tensor product on a 2-category and horizontal composition.

Notice further that an equivalent definition of 2-categories could have been given by just recalling that **Cat** is cartesian closed: a 2-category **B** is then just a **Cat**-enriched category – any $\mathbf{B}(a, b)$ is an object of **Cat**. Following this line, one gets the classical notions of 2-functor and 2-natural transformation, which are just **Cat**-functors and **Cat**-natural transformations. A typical example of a 2-category is **Cat**, where 1-cells are functors and 2-cells are natural transformations. A degenerate example is a special 2-category with only identity 2-cells. At the other extreme, a strict monoidal category is a 2-category with one object.

The following definition describes a particular case of a 2-category (Walters 1981).

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Definition A.2. *A locally posetal-2-category* (*lp*-2-*category* for short) **P** consists of:

(i) a set of objects $Ob(\mathbf{P})$;
(ii) for each pair of objects a and b, a *poset*$(\mathbf{P}(a,b), \leqslant)$ of 1-cells (or morphisms);
(iii) for each triple of objects a, b, and c, a monotonic operation

$$\diamond(a,b,c) : \mathbf{P}(a,b) \times \mathbf{P}(b,c) \rightarrow \mathbf{P}(a,c),$$

called composition of 1-cells, which is associative;
(iv) for each object a, a 1-cell in $\mathbf{P}(a,a)$, called the identity 1-cell and denoted by 1_a, which is a left and right identity for composition of 1-cells.

Fact. Let $(S, \leqslant, \wedge, \bot)$ be a meet-semilattice. It is possible to view S as an lp-2-category **S** as follows:

(i) the set of objects $Ob(\mathbf{S})$ is **S**;
(ii) for each pair of objects x and y, the poset $\mathbf{S}(x,y)$ is given by $\{k | k \in \mathbf{S} \text{ and } k \leqslant x \wedge y\}$;
(iii) for each triple of objects x, y, and z, the composition of 1-cells

$$\mathbf{S}(x,y) \times \mathbf{S}(y,z) \rightarrow \mathbf{S}(x,z),$$

is given by the meet operation $\wedge$;
(iv) for each object x, the identity 1-cell 1_x in $\mathbf{S}(x,x)$ is x itself.

By definition, for each pair of objects x and y we have that: $\mathbf{S}(x,y) = \mathbf{S}(y,x)$.

Definition A.3. Let **P** be an *lp*-2-category. A **P**-*category* (also called *category based on* **P**, or enriched over **P**) **C** is a triple $< C, e, a >$, where C is a set (the set $Ob(\mathbf{C})$ of objects of **C**), and

(i) e is a function, called *extent*, from C to $Ob(\mathbf{P})$, and
(ii) a is a function, called *agreement*, from $C \times C$ to $morph(\mathbf{P})$, satisfying the following properties: for each c, d and f in C,
— $a(c,d)$ is an object of $\mathbf{P}(e(c), e(d))$,
— $a(c,c)$ is the identity 1-cell $1_{e(c)}$, and
— $a(c,d) \diamond a(d,f) \leqslant a(c,f)$.

P is also called the *base* 2-*category* of **C**.

Remarks. For each pair of elements c and d in C, $a(c,d)$ selects a 1-cell of $\mathbf{P}(e(c), e(d))$, say z, such that $z \leqslant e(c) \diamond e(d)$. Further, for each element c, $a(c,c)$ selects the greatest object of $\mathbf{P}(e(c), e(c))$ (which is $e(c)$ itself).

We say that a **P**-category **C** is *symmetric* iff $a(c,d) = a(d,c)$. This notion makes sense when in an *lp*-2-category for each pair of objects x and y, $\mathbf{P}(x,y) = \mathbf{P}(y,x)$.

We say that a symmetric **P**-category is *skeletal* iff for any two objects c, d in C we have that:

$$\text{if } e(c) = e(d) = a(c,d), \text{ then } c = d.$$

https://doi.org/10.1017/S0960129599002935 Published online by Cambridge University Press

Definition A.4. A **P**-functor $f : \mathbf{C}_1 \rightarrow \mathbf{C}_2$ between two **P**-categories $\mathbf{C}_1 =< C_1, e_1, a_1 >$ and $\mathbf{C}_2 =< C_2, e_2, a_2 >$ is a function from C_1 to C_2, also denoted by f, such that:

i) $e_1(C_1) = e_2(f(C_1))$ for any object C_1 in C_1 , and

ii) $a_1(C_1, d_1) \leqslant a_2(f(C_1), f(d_1))$, for any two objects C_1, d_1 in C_1, where $\leqslant$ is in P.

Definition A.5. Given the *lp*-categories $\mathbf{P}_1 = (S_1, \leqslant_1, \diamond_1, \perp_1)$ and $\mathbf{P}_2 = (S_2, \leqslant_2, \diamond_2, \perp_2)$, a *homomorphism* $\phi : \mathbf{P}_1 \rightarrow \mathbf{P}_2$ is given by:

— a function F from $Ob(\mathbf{P}_1)$ to $Ob(\mathbf{P}_2)$,

— for any pair x and y in $Ob(\mathbf{P}_1) \times Ob(\mathbf{P}_1)$ a functor $F_{[x,y]}$ (that is a monotonic function) from $\mathbf{P}_1(x, y)$ to $\mathbf{P}_2(F(x), F(y))$, such that

(i) for any x in $Ob(\mathbf{P}_1)$ $F_{[x,x]}(1_x) = 1_{F(x)}$, and

(ii) for any triple of objects x, y, and z, and
for any pair of 1-cells h, k in $\mathbf{P}_1(x, y) \times \mathbf{P}_1(y, z)$, $F_{[x,y]}(h) \diamond_2 F_{[y,z]}(k) = F_{[x,z]}(h \diamond_1 k)$.

Homomorphism is a special case of the notion of 2-functor.

Bearing in mind that monoidal categories and *lp*-2-categories are particular cases of 2-categories, the two notions of enrichement can be thought of as particular cases of a more general one.

One can easily verify that our trees correspond to symmetric **S**-categories, and morphisms between them to **S**-functors, while the categories of agents are **STrees**-categories.

Acknowledgements

We wish to thank Alberto Pettorossi, who introduced us to the problems addressed in this work and collaborated with us in the initial phase of the developement of the theory. We are deeply indebted to Rocco De Nicola for stimulating conversations, pertinent remarks and comments about the presentation. We would also like to thank our departments for providing us with the necessary facilities.

References

Aceto, L. and Hennessy, H. (1992) Termination, Deadlock and Divergence. *J. of ACM* **39** 147–187.

Bednarczyk, M. A. (1988) *Categories of Asynchronous Systems*, Ph. D. thesis, University of Sussex.

Bénabou, J. (1967) Introduction to Bicategories. *Springer-Verlag Lecture Notes in Mathematics* **47** 1–77.

Benson, D. B. and Tiuryn, J. (1989) Fixed Points in Free Process Algebras – Part I. *Th. Comp. Sci.* **63** 274–294.

Bergstra, J. A. and Klop, J. W. (1985) Algebra of Communicating Processes with Abstraction. *Th. Comp. Sci.* **37** 77–121.

Bergstra, J. A. and Klop, J. W. (1989) Process Theory based on Bisimulation Semantics. *Springer-Verlag Lecture Notes in Computer Science* **354** 50–122.

Bergstra, J. A., Klop, J. W. and Olderog, E.-R. (1987) Failures without Chaos: A new Process Semantics for Fair Abstraction. In: Wirsing, M. (ed.) *Formal Description of Programming Concepts III*, North Holland 77–103.

Betti, R. and Kasangian, S. (1985) Tree Automata and Enriched Category Theory. *Rend. Ist. Mat. Univ. Trieste, XVII* 71–78.

Betti, R., Carboni, A., Street, R. and Walters, R. F. C. (1983) Variations through enrichment. *J. Pure and Applied Algebra* **29** 109–127.

Cardelli, L. (1982) Real Time Agents. Proc. ICALP 1982. *Springer-Verlag Lecture Notes in Computer Science* **140** 94–106.

Castellani, I., Franceschi, P. and Montanari, U. (1983) Labelled Event Structures: A Model for Observable Concurrency. In: Bjørner, D. (ed.) *Proc. IFI TC2 Working Conference on Formal Description of Programming Concepts II, Garmisch*, North Holland 383–400.

Corradini, F., De Nicola, R. and Labella, A. (1995) Fully abstract models for non-deterministic regular expressions. *Springer-Verlag Lecture Notes in Computer Science* **962** 130–144.

D'Agostino, G. (1998) *Modal Logics and non well-founded Set Theories: translation, bisimulation and interpolation*, Ph. D. thesis, Amsterdam.

Degano, P., De Nicola, R. and Montanari, U. (1993) Observation Trees. In: Purushotaman and Amy Zwarico (eds.) *Proc. NAPAW 92*, Springer 103–118.

Degano, P. and Montanari, U. (1987) Concurrent histories: a basis for observing distributed systems. *J. Comp. Syst. Sci.* **34** 422–461.

De Nicola, R. and Labella, A. (1994) A Completeness Theorem for non-deterministic Kleene Algebras. In: Proc. MFCS '94. *Springer-Verlag Lecture Notes in Computer Science* **841** 536–545.

De Nicola, R. and Labella, A. (1998) Tree Morphisms and Bisimulations. *Electr. Notes in Th. Comp. Sci.* **18** (19 pages).

Eilenberg, S. and Kelly, G. M. (1965) Closed Categories. In: *Proc. of the Conference on Categorical Algebra, La Jolla 1965*, Springer 421–562.

Eilenberg, S. (1974) *Automata, Languages and Machines*, Academic Press.

Goguen, J. A. (1974) On Homomorphisms, Correctness, Termination, Unfoldments and equivalence of Flow Diagram Programs. *J. Comp. Syst. Sci.* **8** 333–365.

Gray, J. W. (1966) Fibred and Cofibred Categories. In: *Proc. of the Conference on Categorical Algebra, La Jolla 1965*, Springer 21–83.

Hoare, C. A. R. (1989) *Communicating Sequential Processes*, Prentice Hall.

Kasangian, S. and Labella, A. (1992) Enriched categorical semantics for distributed calculi. *J. of P. and Appl. Algebra* **83** 295–321.

Kasangian, S. and Labella, A. (1992a) On continuous time agents. *Springer-Verlag Lecture Notes in Computer Science* **598** 403–425.

Kasangian, S., Labella, A. and Murphy, D. (1996) Process Synchronization as Fusion. *Appl. Cat. Struct.* **4** 403–421.

Kasangian, S., Labella, A. and Pettorossi, A. (1987) Enriched Categories for Local and Interaction Calculi. *Springer-Verlag Lecture Notes in Computer Science* **283** 57–70.

Kelly, G. M. (1982) *Basic Concepts of Enriched Category Theory*, Cambridge University Press.

Kozen, D. (1991) A Completeness Theorem for Kleene Agebras and the Algebras of Regular Events. *Proc. LICS '91*, IEEE Press 214–225.

Lawvere, F. W. (1973) Metric Spaces, Generalized Logic, and Closed Categories. *Rend. Sem. Mat. Fis. Milano* **43** 135–166.

Mazurkiewicz, A. (1987) Trace theory. *Springer-Verlag Lecture Notes in Computer Science* **255** 279–324.

Milner, R. (1980) A Calculus of Communicating Systems. *Springer-Verlag Lecture Notes in Computer Science* **92**.

Milner, R. (1983) Calculi for Synchrony and Asynchrony. *Th. Comp. Sci.* **25** 267–310.

Milner, R. (1989) *Communication and Concurrency*, Prentice Hall, London.

Nielsen, M., Plotkin, G. and Winskel, G. (1981) Petri Nets, Event Structures, and Domains. *Th. Comp. Sci.* **13** 85–108.

Park, D. (1981) Concurrency and automata on infinite sequences. In: Proc. of Theoretical Computer Science 1981. *Springer-Verlag Lecture Notes in Computer Science* **104** 167–183.

Pratt, V. (1986) Modelling Concurrency with Partial Orders. *Int. J. of Parallel Programming* **15** 33–71.

Reisig, W. (1985) Petri Nets: An Introduction. *EATCS Monographs on Theoretical Computer Science* **4**, Springer.

Rutten, J. (1990) Explicit canonical representative for weak bisimulation equivalence and congruence. Technical Report CS-R9062, Centrum voor Wiskunde en Informatica.

Smith, M. B. and Plotkin, G. D. (1982) The Category-theoretic Solution of Recursive Domain Equation. *SIAM J. of Comp.* **11** 762–783.

Walters, R. F. C. (1981) Sheaves and Cauchy-Complete Categories. *Cahiers de Top. et Géom. Diff.* **22** 283–286.

Winskel, G.(1984) Synchronization Trees. *Th. Comp. Sci.* **34** 32–82.

Winskel, G.(1987) Event Structures. *Springer-Verlag Lecture Notes in Computer Science* **255** 325–392.

Winskel, G. and Nielsen, M. (1995) Models for Concurrency. *Handbook of Logic and Foundations of Computer Science*, Oxford University Press.