On Causality Observed Incrementally, Finally

Gian-Luigi Ferrari
IEI CNR, Via Santa Maria 46, Pisa, Italy

Ugo Montanari
Dipartimento di Informatica, Università di Pisa, Corso Italia 40, Pisa, Italy

Miranda Mowbray
Hewlett-Packard Science Centre, Corso Italia 114, Pisa, Italy

Abstract

One of the main problems of the partial ordering approach to the description of computations of concurrent and distributed systems is that there is no operation of sequential composition for partial orders which preserves all information on causal dependencies (except for the simple case of series parallel pomsets). As a result, truly concurrent behavioural equivalences are defined by giving integral descriptions of computations: they are not obtained by composing the observations of the elementary steps of the computations. In this paper we introduce the algebra of Concatenable Concurrent Histories. A Concatenable Concurrent History describes a computation by means of a partial ordering of events together with information on the initial and final state of the computation. The operation of sequential conposition is one of the primitive operations of this algebra. We introduce a category of transition systems for CCS (called CCS models) and we show that in this category there exists a CCS model which has the structure of Concatenable Concurrent Histories. This object constitutes our model of observations. As a consequence our observations are *incremental*. The observation mechanism is handled by a labelling construction which is an internal operation of the category of models. Furthermore, we give a bisimulation congruence for CCS, via Concatenable Concurrent History observations, which can be characterized in terms of a final universal property in a suitable subcategory of CCS models. This category also provides a framework in which to interpret a Hennessy-Milner style programming logic which describes properties of the observable behaviour of computations. The equivalence naturally induced by the logic coincides with the congruence induced by the final object. This result expresses a sort of duality between semantic analysis based on observations and programming logics. We obtain an analogous result for a weaker congruence which results when internal actions are unobservable.

1 Introduction

Many models for concurrent systems have been proposed in the literature. Much work on the semantics of distributed systems has been based on the *interleaving* approach [Ho 85,Mil 89,BK 84]. In this class of models, a global state is assumed, and the evolution of a system is described in terms of sequences of global states. As a consequence, dealing with local properties is impossible because the corresponding information has been lost in the abstract behaviour. Instead, *true concurrency* or *partial ordering* models [Re 85,NPW 81,DM 87,Pr 86,BC 88] describe the behaviour of distributed systems in terms of the events they may perform, and the constraints on their occurrence: a partial ordering represents the causal dependencies among events, while concurrency is represented by the absence of ordering. True concurrency models provide a more faithful account of distributed computations. They are well suited to handle properties which explicitly refer to the information about distributed activities.

Several efforts have been devoted to the relation between programming logics and observational models of concurrency. As shown by Abramsky [Ab 88], in the interleaving approach this relation is an extension of the classical Stone duality theorem for boolean algebras. Such duality clarifies the relationships between

*Research supported by, and collaboratively carried out at, Hewlett-Packard Laboratories, Pisa Science Centre, Corso Italia 114, 56125 Pisa, Italy.

equivalence classes of computations and properties (described through programming logics) of processes. Presently, it is not clear whether or not this duality holds for the partial ordering approach.

With respect to process description languages, the true concurrency approach has not yet received a completely satisfactory treatment when compared with the results based on interleaving. True concurrency operational semantics have been developed only recently. The basic idea is to provide an interpretation of the language in terms of Petri Nets [DDM 88a,Ol 87,Go 88], Labelled Event Structures [Win 82,DDM 88b], Causal Trees [DD 89], and so on. Since these descriptions are too concrete, certain behavioural equivalences are introduced by extending the techniques introduced within the interleaving framework [DDM 87,vGG 89,RT 88]. In this way, the problem of finding a truly concurrent semantics is reduced to the problem of defining equivalence classes of programs and computations which express particular aspects of system behaviour with respect to certain notions of observation.

One of the drawbacks of truly concurrent semantics is that the research on obtaining logics equipped with proof systems which emphasize the non sequential properties of processes is still missing conclusive achievements. In particular, little is known on the relations (adequacy, expressiveness results) between non interleaving models and logical languages (see [DF 90] for some preliminary results.)

Moreover, in the case of observational semantics (both interleaving and truly concurrent), the standard representatives of the equivalence classes of programs and computations, if any are actually defined, sometimes do not yield minimal realizations, i.e. they do not form a transition system. The minimal realization would represent the most reduced operational semantics with respect to a notion of program transformation which preserves the observable behaviour of programs. Notice that this is a typical situation in automata theory [Gog 72], and in abstract data type specification [GGM 76,Wa 79]. Minimal realizations and the associated transformations are very convenient (think for instance of equivalent circuits in electronics) since they provide an intuitive and suggestive way of handling abstract semantics, and they are quite useful in practice.

The difficulty of having a minimal realization of truly concurrent operational semantics depends on the fact that observable behaviours of machines are inherently incremental: they are obtained by composing the elementary steps of the machines and their observations. On the contrary, truly concurrent behavioural equivalences are defined by observing just the *global* outcome of computations. The main problem is that in the case of partial ordering models there is no obvious operation of sequential composition of partial orders which preserves all the information on causal dependencies.

This paper aims at solving these problems. Our first starting point is the definition of the semantics of process description languages in terms of categories of transition systems with algebraic structure both on states and transitions [FM 90,Fe 90]. In this approach, the observation mechanism of computations is handled by a labelling or typing technique: every computation is labelled (typed) with its observations. In this framework, we consider behavioural equivalences based on the notion of bisimulation [Pa 81]. In [FM 90,Fe 90] it is shown that the *strong observational congruence* [Mil 80] (the simplest bisimulation equivalence) can be characterized in an algebraic way by considering special simplification morphisms which preserve the algebraic structure of states and transitions, the observations *and* the transitions outgoing from any state. It turns out that the strong observational congruence is characterized by a universal property of finality: the terminal object is a transition system whose states and transitions are congruence classes of agents and computations, i.e. a minimal realization.

This schema can be applied only to behavioural equivalences which are at the same time *congruences* and *bisimulations*. For instance this schema does not work in the case of Milner's *Weak Observational Congruence* [Mil 80] (an equivalence which forgets about internal invisible moves) because the weak congruence is not a bisimulation[1]. Bisimulation equivalences which are also congruences can be characterized in a denotational setting [Ab 88]. The second starting point is the notion of *Dynamic Bisimulation* [MS 90], i.e. a bisimulation which tests the observable behaviour of processes also when they are dynamically embedded in the same context. The dynamic bisimulation is the coarsest interleaving weak bisimulation (with the standard operational semantics of CCS) which is also a congruence.

The third starting point is the notion of *Concatenable Processes.* Although Petri Non Sequential Pro-

[1]The states $\alpha.\tau.\beta.nil$ and $\alpha.\beta.nil$ are observationally congruent, where τ is the invisible action, but the states $\tau.\beta.nil$ and $\beta.nil$ they reach after performing an α transition are not. Thus weak observational congruence is not a bisimulation relation. The weak observational congruence for CCS may become a bisimulation by suitably modifying the standard operational semantics of CCS [vG 87].

cesses [GR 83] have information on both causality and distribution, they lack an operation of sequential composition. The problem of having incremental descriptions of Non Sequential Processes has been successfully tackled by Degano, Meseguer and Montanari [DMM 89]. A Petri Net is seen as a graph with a monoidal operation expressing parallel composition of places and transitions. The free category generated by the graph is introduced and certain axioms are used to define a quotient on its morphisms. Indeed, morphisms can be seen as terms of an algebra with two operations, parallel and sequential composition. The congruence classes of these terms can be represented by Concatenable Processes, which are based on Non Sequential Processes but have extra information which allows sequential composition to be defined. However, Concatenable Processes are still unsatisfactory since the Petri Nets they are based on are not labelled by actions, and since they retain information about the intermediate states.

The basic idea of the paper is the definition of an algebraic theory of process description languages (models and logics) where information on causal dependencies and distribution is properly taken into account, and which allows an incremental approach to the description of computations. We take CCS [Mil 80] as a case study.

In our framework, a transition system (model) for the CCS language consists of an equational type algebra [MSS 90]. The algebraic structure on elements of type *state* is given by the language itself; other type elements characterize transitions, and computations. Instead of considering a single model of the language, we consider a collection of such models: each model represents a specific abstract machine (an interpreter) for the language. The collection of models forms a category, where the morphisms preserve the algebraic structure and represent relations between different abstract machines.

CCS models are not necessarily free models of the given presentation: nonfreeness may reflect a particular interpretation of the operations. This allows us to identify a particular model which plays the role of the model of the observations: the interpretation of the operations on computations implements a calculus. By giving different interpretations to the operations we can define several calculi of computations. The possibility of having different interpretations of the operators with a single underlying scheme is one advantage of using the category theoretical framework. Another advantage is that we can describe transformations between interpreters in a smooth way. Because we are interested in a truly concurrent semantics, the calculus of computations will be a calculus of partial orderings.

We introduce the algebra of *Concatenable Concurrent Histories*, which are essentially Concatenable Processes with action labelled events and without information about the intermediate states. The operation of sequential composition of computations is one of the basic operations of the algebra of Concatenable Concurrent Histories. This algebra is closely related to the model of Concurrent Histories, developed by Degano and Montanari [DM 87].

We show that in the category of CCS models there is an object for which the computations have the structure of the algebra of Concatenable Concurrent Histories. This model constitutes our algebra of observations. As a consequence the observations are incremental.

Observing the computations of a CCS model means finding a morphism from it to the model of observations. This construction builds a category whose objects are CCS models with computations labelled by Concatenable Concurrent Histories, and whose morphisms preserve the observations as well as the algebraic structure. The labelling construction is defined in categorical terms: it is an instance of the general *Comma Category* operation [ML 71]. The simplicity and generality of this labelling construction is one of the advantages of using the category theoretic framework.

The category of CCS abstract machines where causality is observed incrementally provides the formal apparatus to introduce and study the features of behavioural equivalences (and congruences) together with the logics which describe properties of computations. In this paper we consider bisimulation-based behavioural equivalences. As in the case of interleaving semantics, we first introduce a bisimulation equivalence over the elements of type *state* of the initial object. We then prove that this equivalence is also a congruence, and we show that it gives rise to a minimal realization. In other words, the set of congruence classes with respect to such a congruence is the set of elements of type *state* of a CCS model, i.e. an abstract machine of the language, which is a terminal object in the appropriate subcategory of CCS models.

To prove this universal property of finality, we take a subcategory, whose objects are the images of the initial object under some fixed type of simplification mappings and whose morphisms are the simplification mappings themselves. The minimal realization (when it exists) is the final object of this subcategory. In this paper the simplification mappings which we consider are *strict transition preserving homomorphisms*, a variant of the transition preserving homomorphisms introduced in [DDM 88a,AD 89,FM 90]. We prove that when the observations are Concatenable Concurrent Histories the final object exists, and, moreover, the unique mapping from the initial to the final object fully characterizes the bisimulation congruence.

In our framework, logics which describe properties of the observable behaviour of computations can be automatically derived by considering *Dynamic Logics* [Ha 84] which are special modal logics where the modalities are parameterized by terms of the calculus of computations[2]. In this paper, we simply introduce a modal logic (in the style of Hennessy-Milner Logic [HM 85]) whose modalities are parameterized by Concatenable Concurrent Histories. We show that our collection of models provides the right framework to give an interpretation of this logic and to understand the relations with the observational semantics. In fact, it turns out that the equivalence induced by the logic coincides with the equivalence induced by the final object.

In the interleaving semantics of CCS, a special action, the τ action, is used to indicate the occurrence of invisible internal operations. Only the proper treatment of τ actions provides us with a semantics of concurrency (as opposed to explicit time). Forgetting τ actions in the observations means performing an abstraction operation. In the standard interleaving approach, *Weak Observational Equivalence* takes care of this abstraction. However, as mentioned before, in this way we get an equivalence which is not a congruence, and therefore we do not have a minimal realization. A minimal realization can be found by considering dynamic bisimulation [MS 90] instead, which is also a congruence.

In our framework the same approach can be applied. Invisible actions can be handled by modifying the observations so that they are forgotten, but it is still possible to distinguish between an idle system and a system performing an invisible move. Also in this case we get both an algebraic and a logical characterization of a weak partial ordering semantics.

2 The Algebra of Concatenable Concurrent Histories

In this section we introduce the algebra of Concatenable Concurrent Histories. For the following definitions we fix two nonintersecting alphabets P, A. Intuitively, P represents the set of process types, while A represents the set of actions.

Definition 1 (*Concurrent Histories*)
Concurrent Histories are labelled partial orders $(V, \leq, \ell)$ *where the labelling function* $\ell : V \rightarrow P \cup A$ *sends the set of maximal and minimal elements to P and the set of other elements to A. Concurrent Histories are considered up to label preserving partial ordering isomorphisms.*□

The elements with labels in P are called *processes*, those with labels in A are called *events*.

Definition 2 (*Label Indexed Ordering Functions*)
Suppose S is a set with a labelling function ℓ: $S \rightarrow P$. *A label indexed ordering function on the labelled set S is a function* α *from S to the set of natural numbers, such that for each* $p \in P$ *the restriction of* α *to the set of elements labelled p is a bijection on the set* $\{1, 2, \ldots, n_p\}$ *where* $n_p = |\{s \in S : \ell(s) = p\}|$.□

Definition 3 (*Concatenatable Concurrent Histories*)
A Concatenable Concurrent History is a triple (h, β, γ) *where* $h = (V, \leq, \ell)$ *is a Concurrent History for which no element is both maximal and minimal, and* β, γ *are label indexed ordering functions on the labelled sets of minimal and maximal elements of V (called* origins *and* destinations*), respectively. Concatenable Concurrent Histories are defined up to isomorphisms of labelled partial orders that preserve the label indexed ordering functions.*□

The introduction of the label indexed ordering functions allows us to discriminate between different maximal elements (and minimal elements) with the same label. Figure 1 illustrates two concatenable concurrent histories ch_1 and ch_2((a) and (b)); the order relation is depicted through its Hasse diagram

[2]Notice that the modal schema (axiom) which corresponds to the notion of incremental description of computations is $[t_1][t_2]\varphi \longrightarrow [t_1; t_2]\varphi$. This feature can be profitably exploited for defining proof systems emphasizing the non sequential aspects of computations. This topic will be subject of further studies.

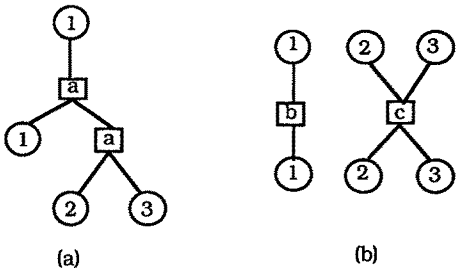

Figure 1: Two Concatenable Concurrent Histories

growing downwards. Processes (resp. events) are represented as circles (boxes): all processes have the same label. Finally, the label indexed ordering functions are represented by positive numbers on processes.

The algebra of Concatenable Concurrent Histories has two operations: parallel composition and sequential composition. Let $ch_1 = (h_1, \beta_1, \gamma_1)$ and $ch_2 = (h_2, \beta_2, \gamma_2)$ be Concatenable Concurrent Histories, where $h_1 = (V_1, \leq_1, \ell_1)$ and $h_2 = (V_2, \leq_2, \ell_2)$. Without loss of generality, V_1 and V_2 are disjoint. Let $\mathrm{Min}(h_1)$, $\mathrm{Max}(h_1)$, $\mathrm{Min}(h_2)$, $\mathrm{Max}(h_2)$ be the origins and destinations of h_1 and h_2, respectively.

Definition 4 (*The Algebra of Concatenable Concurrent Histories*)
The parallel composition $ch_1 \otimes ch_2$ is the Concatenable Concurrent History $((V, \leq, \ell), \beta, \gamma)$ where

- $V = V_1 \cup V_2$
- $\leq$ *is* $\leq_1 \cup \leq_2$
- $\ell(v)$ *is* $\ell_1(v)$ *if* $v \in V_1$ *and is* $\ell_2(v)$ *if* $v \in V_2$
- $\beta(v)$ *is* $\beta_1(v)$ *if* $v \in Min(h_1)$ *and is* $n_1(v) + \beta_2(v)$ *if* $v \in Min(h_2)$, *where* $n_1(v)$ *is the number of origins of* ch_1 *with label equal to* $\ell_2(v)$
- $\gamma(v)$ *is* $\gamma_1(v)$ *if* $v \in Max(h_1)$ *and is* $m_1(v) + \gamma_2(v)$ *if* $v \in Max(h_2)$, *where* $m_1(v)$ *is the number of destinations of* ch_1 *with label equal to* $\ell_2(v)$

The sequential composition $ch_1; ch_2$ is defined if and only if $\ell_1(Max(h_1)) = \ell_2(Min(h_2))$, intended as multisets. In this case the result of the operation is the Concatenable Concurrent History $((V, \leq, \ell)\ \beta, \gamma)$ where

- $V = V_1 \cup V_2 \setminus (Max(h_1) \cup Min(h_2))$
- $\leq$ *is the restriction to* $V \times V$ *of the transitive closure of*
  $\leq_1 \cup \leq_2 \cup \{(w, v) : w \in Max(h_1), v \in Min(h_2), \ell_1(w) = \ell_2(v), \gamma_1(w) = \beta_2(v)\}$
- $\ell(v)$ *is* $\ell_1(v)$ *if* $v \in V_1$ *and is* $\ell_2(v)$ *if* $v \in V_2$
- $\beta(v) = \beta_1(v)$
- $\gamma(v) = \gamma_2(v)$

□

Figure 2 shows the result of the parallel composition $ch_1 \otimes ch_2$ (a), and the result of the sequential composition ch_1 ; ch_2 (b) of the two histories depicted in Figure 1(a) and 1(b). Notice that $\otimes$ is associative but not commutative. In fact, because histories describe the causal relations between events performed during a computation, $\otimes$ should not be commutative, otherwise the operation of sequential composition would not preserve the information on causal dependencies. As an illustration of this fact, let t_μ be the Concatenable Concurrent History with three linearly ordered elements, the middle one of which is labelled μ. Then in the Concatenable Concurrent History $(t_\alpha \otimes t_\beta); (t_\gamma \otimes t_\delta)$ the event labelled α causes the event labelled γ, but in $(t_\beta \otimes t_\alpha); (t_\gamma \otimes t_\delta)$ this is not the case.

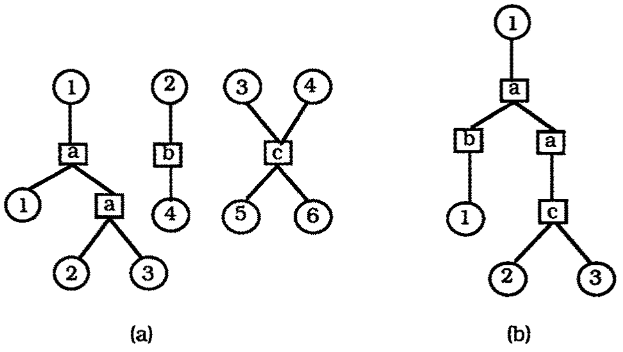

Figure 2: Parallel (a) and sequential composition (b) of the histories in Figure 1(a) and 1(b)

3 Algebraic Models for CCS

In this section we provide an algebraic semantics for CCS in terms of equational type algebras [MSS 90]. Equational type algebras are one sorted algebras enriched with a typing relation which assigns types to elements. The use of these algebras is motivated by the need to represent operational models, i.e. transitions systems, of process description languages.

In the construction of algebraic models for process description languages, one has to face the problem of discriminating between transitions and computations. For instance, in CCS the synchronization can be seen as an operation on transitions but not between computations. Typed algebras allow us to handle such problems. Another problem which can be addressed elegantly by using equational type algebras is the handling of free variables and the construction of recursive processes.

We describe transition systems in terms of equational type algebras in a natural way. The idea is that states of the transition system are elements of the algebra having a certain type. Also, transitions[3] and computations are elements with certain fixed types.

Intuitively, the typing information expresses when an element (state, transition, computation) should be considered *correct*. This is the standard notion of typing in programming languages: only well typed programmes are relevant. We first recall the basic definitions of equational type algebras. See [MSS 90] for a detailed introduction. Let Σ be a one sort signature, a set of operator symbols together with their arity.

Definition 5 (*Equational Type Algebras*)
An equational type algebra A over the signature Σ is a pair $(A, :_A)$, where A is a one-sort total algebra over Σ, and $:_A$ is a binary relation, called the typing relation, on the carrier of A.
A morphism from the equational type algebra A into the equational type algebra B is a mapping h from A to B which respects the operations and the typing, i.e. if σ is an r-ary operator in Σ then $h(\sigma(a_1, \ldots, a_r))$ $= \sigma(h(a_1), \ldots, h(a_r))$, and if $a :_A t$ then $h(a) :_B h(t)$. $\square$

Let Δ be the alphabet for actions, (ranged over by α) and $\overline{\Delta}$ the alphabet of complementary actions (with $\Delta = \overline{\overline{\Delta}}$). Let $\tau \notin \Delta \cup \overline{\Delta}$ be the invisible action, and let $\Lambda = \Delta \cup \overline{\Delta} \cup \{\tau\}$ (ranged over by μ) be the set of actions. We first recall that a CCS expression E has the following syntax,

$$E := nil \mid x \mid \mu.E \mid E\backslash\alpha \mid E[\Phi] \mid E + E \mid E|E \mid rec\, x.E$$

where x is a variable belonging to a set Var of variables, and Φ is a permutation of Λ fixing τ and the operation of complementation. A CCS agent is a CCS expression without free variables. A guarded CCS agent is a CCS agent where each free occurrence of a variable x in $recx.E$ is within a subexpression $\mu.E'$.

[3]The algebraic structure of transitions can be automatically derived by considering the inference rules defining the operational semantics as operations [BC 89,MY 89,FM 90,Fe 90].

A transition system for CCS (which will also be called a CCS model) is modelled as an equational type algebra. The elements of the algebra which are guarded CCS agents have type *state*. CCS expressions may have free and unguarded variables, thus we use (X, Y), where $X, Y \subseteq Var$, to indicate the type of a CCS expression whose free variables are X and whose unguarded variables are Y. Notice that any element with type *state* also has type (ϕ, ϕ), and that if an element has type (X, Y) then $X \subseteq Y$. In the following, generic elements (sometimes called terms) will be denoted by e, elements of type state by u, v, w, and variables by x.

The elements which are transitions (resp. computations) have type *trans* (*comp*). On elements of these types, two special operators ∂_0, ∂_1 are defined. Strictly speaking, these operators are defined on all elements, but $\partial_0(t)$, $\partial_1(t)$ are only typed if t has type *trans* (*comp*). These operators give the source and target states of transitions and computations. As a matter of notation, we will use $t : u \rightarrow v$ as shorthand for "t has type *trans*, $\partial_0(t) = u$, $\partial_1(t) = v$". Similarly we will use $c : u \Rightarrow v$ to indicate that c has type *comp*, $\partial_0(c) = u$, and $\partial_1(c) = v$.

Definition 6 (*CCS Model*)
A CCS model *is an equational type algebra which is a model of the following presentation. The first set of axioms handles standard CCS operators, and gives conditions for an element to have type state.*

$$NIL : (\phi, \phi) \qquad x : (\{x\}, \{x\}) \qquad \frac{e : (X, Y)}{\mu.e : (X, \phi)}$$

$$\frac{e : (X, Y)}{e[\Phi] : (X, Y)} \qquad \frac{e : (X, Y)}{e \backslash \alpha : (X, Y)} \qquad \frac{e_i : (X_i, Y_i), i = 1, 2}{e_1 + e_2 : (\cup_i X_i \cup_i Y_i)}$$

$$\frac{e_i : (X_i, Y_i), i = 1, 2}{e_1 \mid e_2 : (\cup_i X_i, \cup_i Y_i)} \qquad \frac{e : (X, Y), x \notin Y}{rec\, x.e : (X \cap \overline{\{x\}}, Y)} \qquad \frac{u : (\phi, \phi)}{u : state}$$

The next axiom defines the unwinding of recursion[4].

$$\frac{rec\, x.e : state}{recx.e = e[recx.e/x]}$$

The final set of axioms deals with the typing of transitions and computations, and gives the results of the source and target operators explicitly.

$$[\mu, v >: \mu.v \rightarrow v \qquad \frac{t : u \rightarrow v}{t[\Phi] : u[\Phi] \rightarrow v[\Phi]} \qquad \frac{t : u \rightarrow v}{t \backslash \alpha : u \backslash \alpha \rightarrow v \backslash \alpha}$$

$$\frac{t : u \rightarrow v}{t <+ w : u + w \rightarrow v} \qquad \frac{t : u \rightarrow v}{w +> t : w + u \rightarrow v} \qquad \frac{t : u \rightarrow v}{t \rfloor w : u \mid w \rightarrow v \mid w}$$

$$\frac{t : u \rightarrow v}{w \lfloor t : w \mid u \rightarrow w \mid v} \qquad \frac{t_1 : u_1 \rightarrow v_1, t_2 : u_2 \rightarrow v_2}{t_1 \mid t_2 : u_1 \mid u_2 \rightarrow v_1 \mid v_2} \qquad \frac{t : u \rightarrow v}{t : u \Rightarrow v}$$

$$idle(u) : u \Rightarrow u \qquad \frac{c_1 : u \Rightarrow v, c_2 : v \Rightarrow w}{c_1; c_2 : u \Rightarrow w}$$

□

To illustrate the handling of free variables and recursive terms, consider the following elements.

$x : (\{x\}, \{x\})$ *and* $\mu.x : (\{x\}, \phi)$

Applying the inference rules for recursive terms we have that $recx.\mu.x : (\phi, \phi)$. On the other hand, $recx.\mu.x + x$ is defined but not typed: it is unguarded. It should be noticed that the unwinding of recursion can only be applied to guarded variables. For instance, let us consider the deduction

$$\frac{recx.\mu.x : state}{recx.\mu.x = \mu.recx.\mu.x}$$

the guardedness requirement ensures that recursive terms have a unique solution [Mil 89].

[4]Variable replacement could be expressed explicitly in equational type algebras.

Definition 7 *The category CatCCS consists of the CCS models and the morphisms of equational type algebras between them which preserve variables.* □

Proposition 8 *CatCCS has an initial object I.*□

The object I is obtained by free construction with respect to the axioms. It is immediate to prove that the elements of I with type *state* are guarded CCS agents (modulo the equation on recursion).

4 Observing Causality Incrementally

As it stands, the objects of the category CatCCS do not include any mechanism to observe computations, in that the elements of type *comp* represent computations but there is no further abstraction corresponding to what can be detected by an external observer. For instance, in the initial object there is the element $[\alpha, NIL>|\,[\beta, NIL>$ which is not a legitimate computation if we assume the standard synchronization rule of CCS.

In this paper we follow the approach introduced in [Fe 90,FM 90]: the observation mechanism is handled by an operation of labelling (typing) which is internal to the category. We select a specific object of $CatCCS$ to be the model of observations. Defining an observation mechanism corresponds to selecting a morphism from a CCS model to the model of the observations.

Definition 9 (*Observing Computations*)
Let L be any CCS model. Then $CatCCS : L$ is the category whose objects are pairs $(C, \ell : C \to L)$ where C is a CCS model, and ℓ is a CatCCS morphism. The morphisms of $CatCCS : L$ are maps ψ: $(C_1, \ell_1 : C_1 \to L) \to (C_2, \ell_2 : C_2 \to L)$ such that ψ is a CatCCS morphism from C_1 to C_2 and $\ell_2(\psi(c)) = \ell_1(c)$ for all elements c in C_1.□

The CCS model L plays the role of the algebra of observations. The definition above expresses that the morphisms of $CatCCS : L$ are morphisms of $CatCCS$ which preserve the observations. Let ℓ_L denote the unique $CatCCS$ morphism from I to L.

Proposition 10 *$(I,\ \ell_L)$ is the initial object of $CatCCS : L$.* □

As the algebra of observations we will choose a CCS model where the operations are suitably interpreted. More specifically, the model of the observations will include a subset $*$ of elements of type *comp*. A computation is labelled with an element not in $*$ if and only if it is a legitimate computation with respect to the chosen notion of observability. This approach to legitimacy of computations means that we can change the notion of observability just by changing the interpretations of the operators in the algebra of observations. For instance we could interpret $|$ as the CSP parallel operator, or $\backslash$ as hiding. Thus, the synchronization algebra is not forced by the presentation.

We can now define an observation mechanism for CCS which takes distributed and causal information into account. This is done by choosing, as the model of observations, a CCS model H for which the computations which are not in $*$ have the structure of the algebra of Concatenable Concurrent Histories. We first need to introduce some notation.

With CCH we indicate the algebra of Concatenable Concurrent Histories with the alphabet P of processes being a singleton, and the alphabet A of events being the set Λ of actions. We say that an element of CCH is *bipartite* if all its elements are either minimal or maximal. We write $idle_1$ for the bipartite history with just two elements and $idle_n$ for the parallel composition of n copies of this. We write $\wedge_n$ for the bipartite history with $n+1$ elements, n of which are maximal, and $\vee_n$ for the bipartite history with $n+1$ elements, n of which are minimal. We write t_μ for the Concatenable Concurrent History with three linearly ordered elements, where the middle one is labelled μ.

The set of elements of H of type *comp* is the disjoint union of CCH with $*$. There is one element $*_{n,m} : \mathbf{n} \to \mathbf{m}$ of $*$ for each pair of positive integers integers n, m. Elemements of $*$ are absorbent for the operations of parallel and sequential composition of the Algebra of Concatenable Concurrent Histories, in the sense that, given a Concatenable Concurrent History h, $h \oplus *_{n,m}$, $*_{n,m} \oplus h$, $h; *_{n,m}$ and $*_{n,m}; h$ all yield $*_{n',m'}$, for suitable n', m'.

In order to define the CCS model H of causal observations we will give some interpretations of operators in H. The CCS model H is the free CCS model with these operations[5]. We denote the interpretation of a by $[\![a]\!]$.

If e has type (X, Y) then we define the interpretation of e, $[\![e]\!]$, to be $[X, Y, N(e)]$, where $N(e)$ is defined as follows.

- $\mathcal{N}(NIL) = \mathcal{N}(x) = \mathcal{N}(\mu.e) = \mathcal{N}(e + e') = 1$,
- $\mathcal{N}(e[\Phi]) = \mathcal{N}(e \backslash \alpha) = \mathcal{N}(recx.e) = \mathcal{N}(e)$,
- $\mathcal{N}(e \mid e') = \mathcal{N}(e) + \mathcal{N}(e')$.

We use [n] as shorthand for $[\phi, \phi, \mathrm{n}]$, i.e. for denoting the elements of type *state* in H. Intuitively the interpretation of the elements of type *state* detects the number of sequential components which can be considered as autonomous processes. Notice that applying the nondeterministic choice operator gives a *global state*, i.e. a state with just one autonomous component. This assumption corresponds to having a centralized mechanism to deal with non deterministic choices (see [DDM 90,DDM 89] for a deeper discussion on this topic).

The interpretation of operators yielding elements of type *trans* is subject to the following. Transition t stands for an element such that $[\![t]\!] : [\mathrm{n}] \to [\mathrm{m}]$ is not in $*$; we assume also that the interpretation $[\![u]\!]$ of state u is [k]. Elements in $*$ are, as usual, absorbent with respect to all the operations on transitions.

- $[\![[\mu, u >]\!] \;=\; t_\mu; \wedge_k$
- $[\![t \backslash \alpha]\!] \;=\; [\![t]\!]$ if there are no elements of $[\![t]\!]$ labelled α or $\overline{\alpha}$; otherwise $[\![t \backslash \alpha]\!] \;=\; *_{n,m}$
- $[\![t[\Phi]]\!]$ is the history obtained from $[\![t]\!]$ by relabelling each element labelled μ with $\Phi(\mu)$.
- $[\![t < +u]\!] \;=\; [\![u + > t]\!] \;=\; \wedge_n; [\![t]\!]$
- $[\![t \rfloor u]\!] \;=\; [\![t]\!] \otimes idle_k$
- $[\![u \lfloor t]\!] \;=\; idle_k \otimes [\![t]\!]$
- if $[\![t_1]\!] = (\eta_1; (idle_{k1} \otimes t_\lambda); \gamma_1)$ and $[\![t_2]\!] = ((\eta_2; t_{\overline{\lambda}} \otimes idle_{k2}); \gamma_2)$ for some bipartite histories $\eta_2, \gamma_1, \eta_2, \gamma_2$, then $[\![t_1 \mid t_2]\!] \;=\; (\eta_1 \otimes \eta_2); (idle_{k1} \otimes (\vee_2; t_\tau; \wedge_2) \otimes idle_{k2}); (\gamma_1 \otimes \gamma_2)$ Otherwise $[\![t_1 \mid t_2]\!] = *_{n_1+n_2, m_1+m_2}$.

  Finally, let $c_1 : [\mathrm{n}] \Rightarrow [\mathrm{m}]$, $c_2 : [\mathrm{m}] \Rightarrow [\mathrm{p}]$, with $[\![c_1]\!]; [\![c_2]\!] \notin *$.
- $[\![\mathrm{idle}(u)]\!] \;=\; idle_k$
- $[\![c_1; c_2]\!] \;=\; [\![c_1]\!]; [\![c_2]\!]$.

We can comment briefly on the interpretation of the operators. The interpretation of the operator $[\mu, u >$ expresses that after the action μ a *fork* operation is performed, making explicit the distributed structure of state u. Moreover, the non deterministic operation requires first an action of choice between the two alternatives, and then the execution of the chosen alternative. Finally, in the interpretation of the synchronization operator we have adopted a normal form of computations like the one introduced in [GM 90] (see the example below). With these interpretations of the operators, H is indeed a CCS model.

The definition of H implies that $a \in H$ has type b if and only if there is some $a' \in I$ such that a' has type b and $[\![a']\!] = a$ (if b is a type in I then the interpretation of b in H is also called b, because the elements used as types in a CCS model are constants of the algebra).

Proposition 11 *H is a CCS model.* □

Example 12 (*The Synchronization Law*) *Consider the CCS agent $E = (\alpha.nil \mid \beta.nil) \mid ((\overline{\alpha}.nil \mid \delta.nil) + \gamma)$. The synchronization of the transitions labelled α and $\overline{\alpha}$ is represented in H by*

[5]The clauses expressing the interpretation of the operators can be understood as equations of equational type algebras. Thus the CCS model H is the initial algebra of the extended presentation given by adding these equations to the original presentation. E.g. types in H are the same that in I.

$$(\bowtie; (idle_1 \otimes t_\alpha); \bowtie) \mid (\wedge_2; (t_{\overline{\alpha}}; idle_1)) = (\bowtie \otimes \wedge_2); (idle_1 \otimes (\vee_2; t_\tau; \wedge_2) \otimes idle_1); (\bowtie \otimes idle_2)$$

where $\bowtie$ has two origins p_1, and p_2, two destinations q_1, and q_2, and no events, with $p_1 \leq q_2$, $p_2 \leq q_1$, $\beta(p_i) = \gamma(q_i)$, $i = 1, 2$. See Figure 3 for a pictorial representation (the labelled indexed ordering functions are the obvious ones, so we do not represent the numbering on processes).□

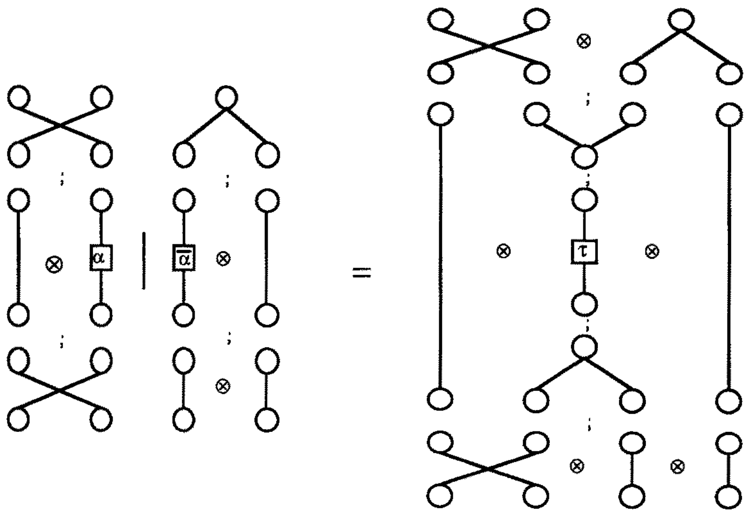

Figure 3: A graphical representation of the observation for the synchronization of transitions in the CCS agent $(\alpha.nil \mid \beta.nil) \mid ((\overline{\alpha}.nil \mid \delta.nil) + \gamma)$

5 Bisimulation Semantics

In this section we introduce a truly concurrent observational semantics for the CCS language by means of the notion of bisimulation. Fix an object $(C, \ell : C \rightarrow H)$ in the category $CatCCS : H$.

Definition 13 (*Bisimulation*)
A bisimulation *on the elements of C with type* state *is an equivalence relation R such that if $u\ R\ u'$ then (i) $\ell(u) = \ell(u')$ and (ii) for every computation $c : u \Rightarrow v$, $c \neq idle(u)$, $\ell(c) \notin *$, there is a computation $c' : u' \Rightarrow v'$, $c' \neq idle(u')$, $\ell(c) = \ell(c')$ and $v\ R\ v'$.* □

The first condition in the definition of a bisimulation ensures that equivalent agents have the same distributed structure, i.e. they are labelled with the same state in H. Notice that for fixed u R u', if there is a computation c which satisfies the hypotheses of condition (ii), then condition (ii) implies condition (i) because $\ell(u') = \ell(\partial_0(c')) = \partial_0(\ell(c')) = \partial_0(\ell(c)) = \ell(\partial_0(c)) = \ell(u)$. In the case when C is the initial CCS model, there is a unique morphism ℓ_H from C to H.

Proposition 14 (*Concurrent History Equivalence*)
There is a unique maximal bisimulation $\sim_H$ on the set of elements of I with type state, *which is the union of all bisimulations.*
Relation $\sim_H$ is an equivalence relation and it is called Concurrent History Equivalence. □

When restricted to sequential agents (agents without the |) the equivalence $\sim_H$ coincides with the strong observational congruence. Milner axiomatization [Mil 89] is consistent and complete for finite sequential agents.

Example 15 (*Result of Observing Distribution and Causality*)
We have that $\alpha.NIL + \alpha.NIL \sim_H \alpha.NIL$, but the two agents $(\alpha.NIL \mid \beta.NIL) + (\alpha.NIL \mid \beta.NIL)$ and $(\alpha.NIL \mid \beta.NIL)$ are not identified because of the global control mechanism for the non deterministic choice. The states $\alpha.NIL \mid NIL$ and $NIL \mid \alpha.NIL$ are not identified, and therefore $\mid$ is not commutative with respect to $\sim_H$. However, it is easy to convince oneself that this must be the case. In fact, assume the two states above are the intermediate states of computations starting from $\beta.\alpha.nil \mid \gamma.nil$ and $\beta.nil \mid \gamma.\alpha.nil$. The identification of the intermediate states (after the parallel execution of the actions β and γ) would imply the impossibility of detecting the correct cause of the action α.
The operator $\mid$ is associative with respect to $\sim_H$, and the operator $+$ is both commutative and associative with respect to $\sim_H$. □

Example 16 (*Expressive Power*)
The two CCS agents $E_1 = \alpha.(\beta.nil + \gamma.nil) + \alpha.nil \mid \beta.nil$, $E_2 = \alpha.(\beta.nil + \gamma.nil) + \alpha.nil \mid \beta.nil + \alpha.\beta.nil$ are indistinguishable by Pomset Bisimulation Equivalence [BC 88]. However, they are distinguished by $\sim_H$. This is because the agent E_2 can perform a computation with observation t_α ending in a state from which a computation with observation t_γ is impossible. The agent E_1 cannot do this because there is no possible computation with observation t_α from the state $\alpha.nil \mid \beta.nil$, although there is a computation with observation $t_\alpha \otimes idle_1$.□

6 Minimal Realization

Concurrent History Equivalence relates elements of type *state* which are indistinguishable in the observational scenario provided by the algebra of Concatenable Concurrent Histories. In this section we show that this equivalence is characterized by an object of the category $CatCCS : H$. In other words, there exists an interpreter of CCS whose states are congruence classes with respect to this equivalence: such an interpreter is a minimal realization. To this aim we consider the subcategory of $CatCCS : H$ of *admissible behaviours*, $Beh_H CCS$, whose objects are CCS models typed on H, and whose morphisms are *strict transition preserving morphisms* (stp morphisms for short).

Definition 17 (*Strict Transition Preserving Homomorphism*)
A CatCCS:H morphism $g : (C_1, \ell_1 : C_1 \to H) \to (C_2, \ell_2 : C_2 \to H)$ is a stp morphism if and only if

(i) g is surjective. Furthermore, if $a_2 \in C_2$ has type b then there is some $a_1 \in C_1$ with type b such that $g(a_1) = a_2$;

(ii) if $c_2 : g(u_1) \Rightarrow v_2$ is a computation C_2 and $\ell_2(t_2) \notin *$, then there is a computation $c_1 : u_1 \Rightarrow v_1$ of C_1 with $g(c_1) = c_2$;

(iii) if c is a computation of C_1 then $g(c) = idle(g(u))$ only if $c = idle(u)$.

□

Definition 18 *(The Category of Admissible Behaviours $Beh_H CCS$)*
The objects of $Beh_H CCS$ are pairs $(C, \ell : C \to H)$ which are images of stp morphisms in CatCCS:H from $(I, \ell_I : I \to H)$. The morphisms of $Beh_H CCS$ are stp morphisms. □

Notice that all morphisms of Beh_HCCS also respect types, e.g. *trans* $\neq$ *state* since this is true for the CCS model H, and H is final in $CatCCS : H$. From now on the pairs like $(C, \ell : C \to H)$ will denote elements of Beh_HCCS. We will prove that Beh_HCCS has a final object.

Theorem 19 *(Minimal Realization Theorem)*
The category $Beh_H CSS$ has a final object, which is a minimal realization of CCS (with observation in H).
Proof *(outline)*
The proofs consists of four steps.

(1) A stp morphism $g : (C, \ell) \to (C', \ell')$ naturally induces a congruence R_g over the elements of (C, ℓ) which behaves like a bisimulation. Two elements are in the same congruence class provided that they have the same image under the morphism g, namely

$aR_gb \iff g(a) = g(b)$.

Condition (i) of the definition of stp morphisms ensures that the relation induced by the morphism is a congruence that preserves the typing. Conditions (ii) and (iii) ensure that this congruence behaves like a bisimulation.

(2) The quotient (Q, ℓ_Q) of (C, ℓ) with respect to the congruence R_g is a CCS model labelled on H. The quotient is defined as follows: let σ be any operation, and let $[a]$ denote the congruence class of a with respect to R_g, i.e. an element of Q. We have:

- *$\sigma([a_1], \dots, [a_r])$ is defined to be $[\sigma(a_1, \dots, a_r)]$.*
- *The element $[a]$ has type b if and only if there is some $a' \in [a]$ with type b.*

The structure of stp morphisms ensures that the operations and the typing are we;ll defined. Therefore Q is a CCS model.
Finally, ℓ_Q is the map satisfying $\ell_Q([a]) = \ell(a)$ for all a. It is immediate to see that this mapping is well defined, and it is a $CatCCS$ morphism from Q to H. This means that (Q, ℓ_Q) is an object of $CatCCS : H$.

(3) The quotient mapping q sending each element to its congruence class under R_g is a stp morphism. In fact it is surjective and respects the typing. The other conditions of the definition of stp morphisms follow immediately from the fact that g is a stp morphism.

(4) The union of all the congruences induced on the initial object (I, ℓ_H) is a congruence which behaves like a bisimulation. The quotient of the initial object with respect to such congruence is a CCS model labelled in H, and the quotient mapping is a stp morphism. Thus, the quotient is the final object of the category $Beh_H CCS$.

□

Let $\approx_H$ be the congruence induced by the final object. We show now that the restriction of $\approx_H$ on the elements of type *state* coincides with the Concurrent History Equivalence.

Theorem 20 (*Characterization Theorem*)
The congruence induced on the elements of type state by the unique stp morphism from (I, ℓ_H) to the final object of $Beh_H CCS$ coincides with $\sim_H$.
Proof *(Outline)*
Recall that guarded CCS agents coincide with the elements of I with type state. The second and third conditions of the definition of stp morphism ensure that the restriction of $\approx_H$ to the elements of type state is a bisimulation.
Conversely, the Concurrent History Bisimulation $\sim_H$ is an equivalence relation which contains pairs of elements of type state. However, it is possible to extend it in an unique way the to obtain a congruence included in $\approx_H$.□

7 Logical Characterization

So far, we have defined an algebraic observational semantics for CCS. In this section, we introduce a (logical) language to express properties of computations, such that the discriminating power of the language is exactly that of the equivalence $\sim_H$, thus reflecting a sort of duality between the two representations.

The language of properties takes the form of a Modal Logic HML(H) in the style of Hennessy-Milner Logic [HM 85] whose modalities are $< h >$, where h is an element of H, h : *comp* and $h \neq *$, i.e. a Concatenable Concurrent History.

Definition 21 ($HML(H)$ *Logic*)
The syntax of the modal logic HML(H) is

$$\psi ::= TRUE \mid [\mathbf{n}] \mid \neg\psi \mid \bigwedge_{j \in J} \psi_j \mid < h > \psi$$

where J is a (possibly infinite) nonempty set of indices, and h is an element of H, $h : comp$ and $h \neq *$.

We define the satisfaction relation $\models$ for HML(H) on the set of elements of type state as follows:
$u \models TRUE$
$u \models [\mathbf{n}]$ *if and only if* $\ell(u) = [\mathbf{n}]$
$u \models \neg\psi$ *if and only if* $u \not\models \psi$
$u \models \bigwedge_{j \in J} \psi_j$ *if and only if* $u \models \psi_j$ *for each* $j \in J$
$u \models < h > \psi$ *if and only if there is a computation* $c : u \Rightarrow v$, $c \neq idle(u)$, *such that* $\ell(c) = h$ *and* $v \models \psi$.□

We comment briefly on the definition of the logic. The family of atomic formulae $[\mathbf{n}]$ is introduced because we need to describe the distributed structure of states, namely a state u satisfies the atomic formula $[\mathbf{n}]$ if and only if it has n autonomous components. In the interleaving case this kind of atomic formula does not provide any discriminating power since a global state is assumed, i.e. the formula $[\mathbf{1}]$ is always satisfied.

The satisfaction relation naturally induces an equivalence relation $\equiv_H$ on the elements of I of type *state*. We say that $v_1 \equiv_H v_2$ if and only if $(v_1 \models \psi) \iff (v_2 \models \psi)$ for all formulae ψ of HML(H).

Theorem 22 (*Logical Characterization*)
The equivalence $\equiv_H$ coincides with $\sim_H$
Proof
Immediate. The proof of the theorem follows the same pattern of the proofs given by Hennessy and Milner in [HM 85].□

8 Forgetting about Internal Moves

The calculus of computations we have given treats all actions as observable. A further abstraction making some actions invisible to the external observer is possible. The standard example of abstraction from invisible actions is provided by Milner's τ action and the *Weak Observational Equivalence* [Mil 80,Mil 89]. The idea is that τ-actions represent internal activities which do not affect the observable behaviour of processes. States (processes) are equivalent provided that they can perform the same *visible* computations, and then reach equivalent states. As a consequence of the abstraction from invisible moves, it might happen that a transition, and a computation performing several internal moves, exhibit the same observable behaviour.

In our framework, the abstraction on silent moves can be obtained by considering a model where the observations of legitimate computations are either idle or have some observable action. In the case of a truly concurrent observational semantics the model of the legitimate, non-idle observations has the form of the algebra of Concatenable Concurrent Histories with no action labelled with τ. We denote by H_W this model of observations. The CCS model H_W can be obtained from the CCS model H by imposing the equation[6]

$$t_\tau = idle_1.$$

This equation clearly expresses that τ moves are invisible in the model of the observations H_W.

As a further step in our construction, we build a category whose objects are CCS models labelled (typed) over this weaker calculus of computations, and whose morphisms respect the observations.

A (weak) bisimulation semantics is immediately obtained. This observational semantics abstracts from τ actions but it is still possible to distinguish between an idle transition and a computation performing an invisible move. We indicate with $\sim_{H_W}$ the maximal bisimulation on the elements of type *state* of the initial object I.

We then consider the categort Beh_{H_W}CCS of admissible behaviours. Notice that because stp morphisms are strict on identities, the simplification mapping is able to distinguish between an idle move and a computation performing an invisible move.

Also in this case, we prove the characterization theorems.

Theorem 23 (*Minimal Realization for Weak Observations*)
$Beh_{H_W}CCS$ has a final object, which is a minimal realization of CCS (with observation in H_W) □

[6]Recall that with t_μ we indicate a Concatenable Concurrent History with three linearly ordered elements, where the middle one is labelled with μ.

Theorem 24 *(Characterization)*
The congruence $\approx_{H_W}$ which is induced on the elements of type state by the unique stp morphism from initial to the final object coincides with $\sim_{H_W}$.□

As in the case of the CCS model H, we introduce the modal logic HML(H_W) where modalities are parameterized with elements of type *comp* of H_W wwhich are not in $*$. We indicate with $\equiv_{H_W}$ the equivalence induced by the logic. The logical characterization theorem still holds.

Theorem 25 *(Logical Characterization)*
The equivalence relation $\equiv_{H_W}$ generated by HML(H_W) coincides with $\sim_{H_W}$.□

When restricted to sequential agents, this weak congruence coincides with the greatest dynamic bisimulation [MS 90]. In particular, for finite sequential agents the axiomatization consisting of the axioms for the strong observational congruence and of the second and third of Milner's τ laws [7] is consistent and complete.

Example 26 *As an example of the congruence induced by the final object of the category $Beh_{H_W}CCS$, we have that the agents $E + \tau.E$ and $\tau.E$ are identified (E is any agent). Notice that this is Milner's second τ-law. Another law which holds is that $(\tau.E_1) \mid E_2$ and $E_1 \mid (\tau.E_2)$ (with both E_1 and E_2 initial states observed as [1]) are identified. The intuitive idea is that the observer is able to detect that the internal move has taken place, but it is not able to detect the location of such a move. The distribution of initial states is still observed, so that for instance $\tau.(\alpha.nil \mid \beta.nil)$ and $(\alpha.nil \mid \beta.nil)$ are not identified. Finally, the CCS agents $\tau.\tau.\alpha.nil$ and $\tau.\alpha.nil$ are not identified, because the τ move of the first agent should be simulated by the second agent staying idle, which is not permitted.*□

9 Conclusions

We have introduced a theory for CCS (algebraic models and logics), where causal dependencies and distribution are properly taken into account. We have shown that observational models can be equipped with truly concurrent observations which are incremental, and that behavioural congruences can be characterized both by considering special simplification mappings, (from which we have constructed a minimal realization of the theory) and by considering the equivalence induced by a modal logic (from which we have a language of properties). This solves the problem set out in the introduction of finding an operational semantics which takes into account causal dependencies between events, is incremental, has a minimal realization and an associated logic.

We plan to extend the results of this paper, giving sufficient conditions for a category of observational models of a process description language to yield observational equivalence which can characterized by a minimal realization.

References

[Ab 88] Abramsky, S., *A Domain Equation for Bisimulation*, Internal report, Dept. Computing, Imperial College

[AD 89] Arnold, A., Dicky, A., *An Algebraic Characterization of Transition System Equivalences,* Information and Computation **82**, 1989.

[BC 88] Boudol, G., Castellani, I., *Concurrency and Atomicity,* Theoretical Computer Science **59**, 1988.

[BC 89] Boudol, G., Castellani, I., *Permutations of Transitions: An Event Structure Semantics for CCS,* In Proc. REX School, Workshop on Linear Time, Branching Time, and Partial Orders in Logics and Models for Concurrency, LNCS **354**,1989.

[BK 84] Bergstra, J., Klop, W., *Process Algebra for Synchronous Communication,* Information and Control **60**, 1984.

[7] If one insists that also the first τ law holds, the resulting congruence (i.e. the weak observational congruence), as observed above, is not a bisimulation and the characterization by finality does not hold.

[DD 89] Darondeau, P., Degano, P. *Causal Trees*, In Proc. ICALP 89, LNCS **372**, 1989.

[DDM 87] Degano, P. De Nicola, R., Montanari, U. *Observational Equivalences for Concurrency Models*, in Formal Description of Programming Concepts III, M. Wirsing, ed., North Holland, 1987.

[DDM 88a] Degano, P. De Nicola, R., Montanari, U. *A Distributed Operational Semantics for CCS Based on Condition/Event Systems.* Acta Informatica, **26**, 1988.

[DDM 88b] Degano, P. De Nicola, R., Montanari, U. *On the Consistency of Truly Concurrent Operational and Denotational Semantics.* Proc. LICS 1988.

[DDM 89] Degano, P. De Nicola, R., Montanari, U. *Partial Ordering Descriptions and Observations of Nondeterministic Concurrent Processes*, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS **354**, 1987.

[DDM 90] Degano, P. De Nicola, R., Montanari, U. *A Partial Ordering Semantics for CCS*, Theoretical Computer Science **75**, 1990.

[DMM 89] Degano, P., Meseguer, J., Montanari, U., *Axiomatizing Net Computations and Processes,* in Proc. Logics in Computer Science 89, 1989.

[DM 87] Degano, P., Montanari, U., *Concurrent Histories: A Basis for Observing Distributed Systems,* Journal of Computer and System Sciences **34**, 1987.

[DF 90] De Nicola, R. Ferrari, G. *Observational Logics and Concurrency Models*, Proc 10th Conf. FST and TCS, LNCS **472**, 1990

[Fe 90] Ferrari, G. *Unifying Models of Concurrency*, PhD Thesis, TD/4-90, Dipartimento di Informatica, Univ. Pisa, 1990.

[FM 90] Ferrari, G., Montanari, U., *Towards the Unification of Models for Concurrency,* Proc. CAAP'90, LNCS **431**, 1990.

[Go 88] Goltz, U. *On Representing CCS Programs by Finite Petri Nets*, Proc. MFCS 88, LNCS **324**, 1988.

[Gog 72] Goguen, J. *Realization is Universal*, Math. Systems Theory, **6**, 1972.

[GGM 76] Giarratana, V., Gimona, F., Montanari, U. *Observability Concepts in Abstract Data Type Specifications*, in Proc. MFCS 76, LNCS **45**, 1976.

[GM 90] Gorrieri, R., Montanari, U., *SCONE: a Simple Calculus of Nets,* In Proc. CONCUR'90, LNCS **458**, 1990.

[GR 83] Goltz, U., Reisig, W., *The Non Sequential Behaviour of Petri Nets,* Information and Computation **57**, 1983.

[Ha 84] Harel, D. *Dynamic Logics*, in Handbook of Philosophical Logics, Vol II, Gabbay-Guenthner, eds., Reidel 1984.

[Ho 85] Hoare, C.A.R., *Communicating Sequential Processes,* Prentice Hall, 1985.

[HM 85] Hennessy, M., Milner, R., *Algebraic Laws for Nondeterminism and Concurrency,* Journal of ACM **32** 1985.

[Mil 80] Milner, R., *A Calculus of Communicating Systems,* LNCS **92**, 1980.

[Mil 89] Milner, R., *Communication and Concurrency,* Prentice Hall, 1989.

[ML 71] Mac Lane, S., *Categories for the Working Mathematician,* Springer-Verlag 1971.

[MS 90] Montanari, U., Sassone, V., *Dynamic Bisimulation,* Technical Report TR-13/90, Dipartimento di Informatica, Univ. Pisa, 1990.

[MSS 90] Manca, V., Salibra, A., Scollo, G., *Equational Type Logic*, Theoretical Computer Science, **77**, 1990.

[MY 89] Montanari, U., Yankelevich, D., *An Algebraic View of Interleaving and Distributed Operational Semantics,* in Proc. Third Symposium on Category Theory and Computer Science, LNCS **389**, 1989.

[Pa 81] Park, D., *Concurrency and Automata on Infinite Sequences,* in Proc. GI, LNCS **104**, 1981.

[NPW 81] Nielsen, M., Plotkin, G. Winskel, G. *Petri Nets, Event Structures and Domains.* Theoretical Computer Science, **13**, 1981.

[Ol 87] Olderog, E.-R., *Operational Petri Nets Semantics for CCSP,* in Advances in Petri Nets 87, LNCS **266**, 1987.

[Pr 86] Pratt, V., *Modelling Concurrency with Partial Orders,* Int. Journal of Parallel Programming, **15**, 1986.

[RT 88] Rabinovich, A. Trakhtenbrot, B. *Behaviour Structures and Nets,* Fundamenta Informaticae XI, 1988.

[Re 85] Reisig, W. *Petri Nets: An Introduction,* EATCS Monograph, Springer, 1985.

[vG 87] van Glabbeek, R. *The Approximation Induction Principle in Process Alagebras,* Proc. STACS'87, LNCS **397**, 1987.

[vGG 89] van Glabbeek, R., Goltz, U., *Equivalence Notions for Concurrent Systems and Refinement of Actions,* Proc. MFCS '89, LNCS **397**, 1989.

[Wa 79] Wand, M., *Final Algebras Semantics and Data Type Extension,* Journal of Comp. and System Science, **19**, 1979.

[Win 82] Winskel, G., *Event Structures Semantics for CCS and Related Languages,* in Proc. ICALP 82, LNCS **140**, 1982.