In proceedings of MFCS '91, LNCS 520, Springer-Verlag, 1991

CCS DYNAMIC BISIMULATION is PROGRESSING*

Ugo Montanari and Vladimiro Sassone
Dipartimento di Informatica – Università di Pisa
Corso Italia 40 - 56100 - Pisa - Italy
E-MAIL:{ugo,vladi}@dipisa.di.unipi.it

Abstract

Weak Observational Congruence (woc) defined on CCS agents is not a bisimulation since it does not require two states reached by bisimilar computations of woc agents to be still woc, e.g. $\alpha.\tau.\beta.nil$ and $\alpha.\beta.nil$ are woc but $\tau.\beta.nil$ and $\beta.nil$ are not. This fact prevents us from characterizing CCS semantics (when τ is considered invisible) as a final algebra, since the semantic function would induce an equivalence over the agents that is both a congruence and a bisimulation.

In the paper we introduce a new behavioural equivalence for CCS agents, which is the coarsest among those bisimulations which are also congruences. We call it Dynamic Observational Congruence because it expresses a natural notion of equivalence for concurrent systems required to simulate each other in the presence of dynamic, i.e. run time, (re)configurations. We provide an algebraic characterization of Dynamic Congruence in terms of a universal property of finality.

Furthermore we introduce Progressing Bisimulation, which forces processes to simulate each other performing explicit steps. We provide an algebraic characterization of it in terms of finality, two characterizations via modal logic in the style of HML, and a complete axiomatization for finite agents.

Finally, we prove that Dynamic Congruence and Progressing Bisimulation coincide for CCS agents. Thus the title of the paper.

1 Introduction

Understanding concurrent systems is difficult, since many of our intuitions about sequential systems cannot be extended to concurrent and distributed systems. In particular, there is no prevalent, notion of system behaviour on which semantic constructions can be based.

Milner's *Calculus of Communicating Systems* (CCS) ([Mil80], [Mil89]) can be considered the touchstone of process description languages. Its semantics is given in two steps. First, a *Labelled Transition Systems* (LTS), which constitutes the abstract machine (the interpreter) of the language, is defined in the style of Plotkin's *Structured Operational Semantics* (SOS) ([Plo81]). Then behavioural equivalences are introduced.

A large number of such equivalences have been proposed. Several properties are interesting in the analysis of concurrent systems, and each definition stresses a particular aspect of systems behaviour. For instance, if we are interested only in the actions performed by a system, we consider a simple trace equivalence; otherwise, if we allow the possibility of replacing a subsystem by an equivalent one, we must define an equivalence which is a *congruence* with respect to language constructs. Moreover, if we follow the *interleaving* approach ([Mil80], [Mil89]), i.e. if we express concurrency of actions by saying that they may be executed in any order, then we will choose to observe *sequences* of actions. In a *truly concurrent* approach, on the other hand, ([Pet62], [NPW81], [Pra86], [DDM90]) we may want to observe *partial orderings* of actions. For an overview and a comparison of many behavioural equivalences see [De 87] or [GvG89].

Among the equivalences proposed in the literature, we consider those based on *bisimulation* ([Mil80], [Par81], [vGW89]). Two processes are equivalent if they not only produce the same observations, but also reach equivalent states afterwards and, in the case of *Branching Bisimulation*, pass only through equivalent intermediate states. The advantages of those equivalences, besides their operational suggestiveness, are the existence of simple axiomatizations, the elegance of the proofs and their relationship with

*Research supported in part by **HEWLETT–PACKARD Laboratories**, Pisa Science Center.

equivalences based on logics ([Mil89]), on denotational semantics ([Abr88]) and on algebraic techniques ([BBS88], [Acz87]).

Ferrari and Montanari ([FM90]) define *Strong Observational Congruence*, the simplest equivalence based on bisimulation, in an algebraic way. They see the CCS transition system as equipped with an algebraic structure on both states (the operations of the language) and transitions (an operation for every SOS inference rule). Furthermore they define a collection (in fact a subcategory) of such transition systems, where the operations are not necessarily free and where morphisms relating two transition systems are *transition preserving*, i.e. they define simplification mappings which respect, besides operations on both nodes and arcs and besides labels (including τ's) on arcs, the transitions outgoing from any state. This subcategory has an initial and a final element, and the unique morphism from the former to the latter defines the coarsest equivalence on agents that is both a *congruence* and a *strong bisimulation*, i.e. *Strong Observational Congruence*.

A similar construction can be repeated by mapping computations instead of transitions, and disregarding τ's. We obtain the coarsest equivalence that is both a *congruence* and a *weak bisimulation*, but this equivalence is *not* the *Weak Observational Congruence*, since the latter is not a weak bisimulation. Actually, Van Glabbeek ([vG87]) shows that Weak Observational Congruence is a bisimulation, but the operational semantics of CCS he assumes is not the usual one, e.g. $\alpha.\beta \xrightarrow{\alpha} \tau.\beta$. From these facts originated the idea of the new behavioural equivalence introduced in this paper.

The basic idea of *dynamic bisimulation* is to allow at every step of bisimulation not only the execution of an action (or a sequence of actions), but also the embedding of the two agents under measurement within the same, but otherwise arbitrary, context.

Conceptually, bisimulation can be understood as a kind of *game*, where two programs try in turn to match each other's moves. When a move consists of performing some computational steps and matching a move means to produce the same *observable* behaviour, then we obtain the notion of *observational equivalence*. This definition is independent of the particular observable behaviour (τ observable or not, sequences or partial orderings of actions, etc.), and it can be proved under very mild conditions that the maximal bisimulation relation *always* exists and is an equivalence ([MS89]).

Instead of programs just being able to perform computational steps, we might consider *modular* (i.e. compositional) *software systems* which are statically configured at time zero. In our functional setting, this means to start the game by applying an arbitrary context to both agents. The resulting semantic notion is Milner's *Observational Congruence*.

Finally we may allow *dynamic reconfiguration*: at any instant in time the structure of both software systems may be modified, but in the same way; i.e. a context can be applied to both agents. In this way we obtain our new notion of *dynamic bisimulation*, and the resulting semantic equivalence is called *Dynamic Observational Congruence*. Of course the *bisimulation game* is not just an academic fancy but is motivated by practical considerations: equivalent (in the various senses discussed above) modules can actually replace each other consistently in any real system, guaranteeing *software modularity* and *reusability*. In particular, the issue of dynamic reconfiguration is relevant for *system programming* and for applications like *software development*, where executing new programs is essential, and like *industrial automation*, where halting execution may be extremely costly.

In this paper we show that *Dynamic Observational Congruence* is the *coarsest* bisimulation which is also a congruence, and thus it is algebraically characterized by the finality construction in the style of [FM90] outlined above.

Furthermore we introduce a new observational equivalence, *Progressing Bisimulation*, between states of a labelled transition system with a distinct action τ. The basic idea underlying Progressing Bisimulation is to force programs in the bisimulation game to match moves with explicit moves. This also justifies its name.

For Progressing Bisimulation we give an *algebraic characterization* in the category of labelled transition systems and two *modal logics* in the style of HML, one in which the modal operators may include τ's, and their meaning is that at least those τ's must appear in the models, the other in which the satisfaction relation forces agents to move. Then we provide a *complete axiomatization* for states with finite computations, consisting of the axioms for Strong Observational Congruence and of two of the three Milner's τ-laws (of course $\alpha.\tau.P = \alpha.P$ is no longer true).

Finally, we show that on the CCS transition system Progressing Bisimulation coincides with Dynamic Congruence. That gives all the characterizations above to Dynamic Congruence and gives meaning to

the paper's title.

This presentation stresses the fact that we are in presence of two distinct, general concepts, which, in the case of CCS, coincide: *Dynamic Congruence*, which makes sense on the LTS of every language and has a nice algebraic characterization and *Progressing Bisimulation*, which makes sense on every LTS with a distinct action τ and has algebraic, logical and axiomatic characterizations.

The paper is organized as follows.

In section 2 we recall the basic concepts of CCS ([Mil80], [Mil89]). Section 3 provides the *operational definition* and an *algebraic characterization* of Dynamic Observational Congruence. The Progressing Bisimulation and its *algebraic*, *logical* and *axiomatic* characterizations are introduced in section 4. Finally, in section 5 we show that Dynamic Congruence and Progressing Bisimulation *coincide* in the CCS transition system, thus obtaining a full characterization of Dynamic Congruence.

In the paper, we follow the notations and definitions in the references, thus the expert reader can safely skip section 2. For space saving, proofs are only sketched: the full version can be found in [MS90].

2 Calculus of Communicating Systems

In this section we recall the basic definitions of Milner's *Calculus of Communicating Systems* (CCS). For a full introduction however, the reader is referred to [Mil80] and [Mil89].

Let $\Delta = \{\alpha, \beta, \gamma, \ldots\}$ be a fixed set of actions, and let $\overline{\Delta} = \{\overline{\alpha} | \alpha \in \Delta\}$ be the set of complementary actions ($\bar{}$ being the operation of complementation). $\Lambda = \Delta \cup \overline{\Delta}$ (ranged over by λ) is the set of *visible actions*. Let $\tau \notin \Lambda$ be the *invisible action*; $\Lambda \cup \{\tau\}$ is ranged over by μ.

Definition 2.1 *(CCS Expressions and Agents)*
The syntax of CCS expressions is defined as follows:

$$E ::= x \mid nil \mid \mu.E \mid E \setminus \alpha \mid E[\Phi] \mid E + E \mid E|E \mid recx.E$$

where x is a variable, and Φ is a permutation of $\Lambda \cup \{\tau\}$ preserving τ and $\bar{}$. CCS agents *(ranged over by $P, Q, \ldots$) are closed CCS expressions, i.e. expressions without free variables.* □

The operational semantics of CCS is defined in terms of *labelled transition systems* ([Kel76]) in which the states are CCS expressions and the transition relation is defined by axioms and inference rules driven by the syntactic structure of expressions.

Definition 2.2 *(CCS Transition Relation)*
The CCS transition relation $\xrightarrow{\mu}$ is defined by the following inference rules.

Act) $\mu.E \xrightarrow{\mu} E$

Res) $\dfrac{E_1 \xrightarrow{\mu} E_2}{E_1 \setminus \alpha \xrightarrow{\mu} E_2 \setminus \alpha} \quad \mu \notin \{\alpha, \overline{\alpha}\}$

Rel) $\dfrac{E_1 \xrightarrow{\mu} E_2}{E_1[\Phi] \xrightarrow{\Phi(\mu)} E_2[\Phi]}$

Sum) $\dfrac{E_1 \xrightarrow{\mu} E_2}{E_1 + E \xrightarrow{\mu} E_2 \text{ and } E + E_1 \xrightarrow{\mu} E_2}$

Com1) $\dfrac{E_1 \xrightarrow{\mu} E_2}{E_1|E \xrightarrow{\mu} E_2|E \text{ and } E|E_1 \xrightarrow{\mu} E|E_2}$

Com2) $\dfrac{E_1 \xrightarrow{\lambda} F_1 \text{ and } E_2 \xrightarrow{\overline{\lambda}} F_2}{E_1|E_2 \xrightarrow{\tau} F_1|F_2}$

Rec) $\dfrac{E_1[recx.E_1/x] \xrightarrow{\mu} E_2}{recx.E_1 \xrightarrow{\mu} E_2}$

□

The transition $P \xrightarrow{\mu} Q$ expresses that the agent P may evolve to become the agent Q through the action μ, μ being either a stimulus from the environment or the internal action τ which is independent from the environment. Computations are usually described by *multistep derivation* relations derived from $\xrightarrow{\mu}$. The relations we will need in the following are: $\stackrel{\epsilon}{\Longrightarrow}$ defined as $(\xrightarrow{\tau})^*$, where ϵ is the null string and * the transitive closure of relations; $\stackrel{\mu}{\Longrightarrow} = \stackrel{\epsilon}{\Longrightarrow}\xrightarrow{\mu}\stackrel{\epsilon}{\Longrightarrow}$, where $\mu \in \Lambda \cup \{\tau\}$; $\stackrel{t}{\Longrightarrow} = \stackrel{\mu_1}{\Longrightarrow}\stackrel{\mu_2}{\Longrightarrow}\cdots\stackrel{\mu_n}{\Longrightarrow}$, where $t = \mu_1\mu_2\ldots\mu_n \in (\Lambda \cup \{\tau\})^*$ and $\stackrel{s}{\Longrightarrow} = \stackrel{\lambda_1}{\Longrightarrow}\stackrel{\lambda_2}{\Longrightarrow}\cdots\stackrel{\lambda_n}{\Longrightarrow}$, where $s = \lambda_1\lambda_2\ldots\lambda_n \in \Lambda^*$.

The semantics given by labelled transition systems is too concrete: the addition of *behavioural equivalence* equates those processes which cannot be distinguished by any external observer of their behaviour. Park's notion of *bisimulation* ([Par81]) has become the standard device for defining behavioural equivalences.

Definition 2.3 *(Strong Bisimulation)*
Let $\mathcal{R}$ be a binary relation over CCS agents. Then Ψ, a transformation of relations, is defined by

$(P,Q) \in \Psi(\mathcal{R})$ *if and only if* $\forall \mu \in \Lambda \cup \{\tau\}$*:*
- *whenever* $P \xrightarrow{\mu} P'$ *there exists* Q' *s.t.* $Q \xrightarrow{\mu} Q'$ *and* $(P',Q') \in \mathcal{R}$*;*
- *whenever* $Q \xrightarrow{\mu} Q'$ *there exists* P' *s.t.* $P \xrightarrow{\mu} P'$ *and* $(P',Q') \in \mathcal{R}$*.*

A relation $\mathcal{R}$ is called strong bisimulation *if and only if $\mathcal{R} \subseteq \Psi(\mathcal{R})$.*
The relation $\sim = \cup\{\mathcal{R} | \mathcal{R} \subseteq \Psi(\mathcal{R})\}$ is called Strong Observational Equivalence. □

Since Ψ is a monotone function, $\sim$ is the largest strong bisimulation. Moreover, it is an equivalence relation over CCS agents.

Strong Observational Equivalence can be extended to CCS expressions by saying that two expressions are strongly equivalent if all the agents obtained by binding their free variables to CCS agents are strongly equivalent.

However, definition 2.3 does not consider τ actions as special actions representing the occurrence of invisible internal moves. If we take into account the special status of τ actions, agents are equivalent if they can perform the same sequences of visible actions and then reach equivalent states. The notion of *Weak Observational Equivalence* implements this kind of abstraction.

Weak Equivalence, written $\approx$, is defined by introducing a transformation Φ, obtained from the definition of Ψ by replacing $\xrightarrow{\mu}$ by $\stackrel{s}{\Longrightarrow}$, and taking its greatest fixed point, i.e. $\approx = \cup\{\mathcal{R} | \mathcal{R} \subseteq \Phi(\mathcal{R})\}$. Relation $\approx$ is the largest weak bisimulation, and it is an equivalence relation. It is extended to CCS expressions in the same way Strong Equivalence is.

An equivalence ρ is called *congruence* with respect to an operator f, if it is respected by the operator, i.e. $x \rho y$ implies $f(x) \rho f(y)$. The equivalences which are congruences with respect to all the operators of the language are very important: they provide algebras in which equality is mantained in every context, a property that can be exploited by algebraic techniques.

Formally, in our framework, a context $\mathcal{C}[\ ]$ is a CCS expression without free variables and with exactly one "hole" to be filled by a CCS agent.

Relation $\sim$ is a congruence with respect all CCS operators, that is $E \sim F$ implies $\mathcal{C}[E] \sim \mathcal{C}[F]$ for each context $\mathcal{C}[\ ]$, but it is well known that Weak Observational Equivalence is not a congruence. Indeed, we have $\tau.E \approx E$ but in general it is false that $\tau.E + E' \approx E + E'$, e.g. $\tau.\alpha.nil \approx \alpha.nil$ but $\beta.nil + \alpha.nil \not\approx \beta.nil + \tau.\alpha.nil$ because the first agent provides α and β as alternative actions, while the second agent may autonomously discard the β alternative by simply performing a τ action.

The largest congruence contained in $\approx$ is Milner's Weak Observational Congruence, written $\approx^c$ and defined by $P \approx^c Q$ if and only if for any context $\mathcal{C}[\ ]$, $\mathcal{C}[P] \approx \mathcal{C}[Q]$.

Weak Observational Congruence has a main drawback: it is *not* a bisimulation. As an example consider the weakly congruent agents $\alpha.\tau.nil$ and $\alpha.nil$. When $\alpha.\tau.nil$ performs an α action becoming the agent $\tau.nil$, $\alpha.nil$ can only perform an α action becoming nil: clearly $\tau.nil$ and nil are not weakly congruent but only weakly equivalent. Our definition of Dynamic Observational Congruence remedies this situation.

3 Dynamic Observational Congruence

In this section we introduce *Dynamic Bisimulation* by giving its *operational definition* and its *algebraic characterization* in the style of [FM90].

The definition is given for CCS, but it can be given for any labelled transition system in which the concept of context makes sense, in particular for the LTS corresponding to any language.

3.1 Operational definition

We want to find the coarsest bisimulation which is also a congruence. Let $\mathcal{B}$ be the set of (weak) bisimulations and $\mathcal{C}$ be the set of congruences.

Definition 3.1 *(Dynamic Bisimulation)*
Let $\mathcal{R}$ be a binary relation over CCS agents. Then Φ_d, a transformation of relations, is defined as follows:

$(P,Q) \in \Phi_d(\mathcal{R})$ *if and only if* $\forall s \in \Lambda^*$ *and* $\forall \mathcal{C}[\,]$*:*
- *whenever* $\mathcal{C}[P] \stackrel{s}{\Longrightarrow} P'$ *there exists* Q' *s.t.* $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$ *and* $(P',Q') \in \mathcal{R}$*;*
- *whenever* $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$ *there exists* P' *s.t.* $\mathcal{C}[P] \stackrel{s}{\Longrightarrow} P'$ *and* $(P',Q') \in \mathcal{R}$*.*

A relation $\mathcal{R}$ is called dynamic bisimulation *if and only if $\mathcal{R} \subseteq \Phi_d(\mathcal{R})$.*
The relation $\approx^d = \cup\{\mathcal{R} | \mathcal{R} \subseteq \Phi_d(\mathcal{R})\}$ is called Dynamic Observational Equivalence. □

Relation $\approx^d$ is a dynamic bisimulation and it is an equivalence relation. Just as Strong and Weak Equivalence, it is extended to CCS expressions. In the following, whenever it makes sense, we will consider only CCS agents: obviously, results hold also for CCS expressions, by definition of the equivalences on them.
We show now that $\approx^d$ is the coarsest bisimulation which is also a congruence.

Lemma 3.2 *(Dynamic Bisimulations are Weak Bisimulations)*
$\mathcal{R} \subseteq \Phi_d(\mathcal{R})$ implies $\mathcal{R} \subseteq \Phi(\mathcal{R})$, where Φ is the function defining Weak Observational Equivalence.
Proof. Directly from the definitions of Φ and Φ_d (fixing the context $\mathcal{C}[\,] \equiv x$). □

As a corollary to the previous lemma, we obtain that $\approx^d \subseteq \approx$, i.e. Dynamic Equivalence refines Weak Observational Equivalence. However, the reverse inclusion does not hold as $P \approx \tau.P$ while $P \not\approx^d \tau.P$.

Lemma 3.3 *(Dynamic Bisimulations are Congruences)*
Let $\mathcal{R} \subseteq \Phi_d(\mathcal{R})$. Then $P\mathcal{R}Q$ if and only if $\mathcal{C}[P]\mathcal{R}\mathcal{C}[Q]$ for each context $\mathcal{C}[\,]$.
Proof. ($\Rightarrow$) If there were $\mathcal{C}[\,]$ such that $(\mathcal{C}[P], \mathcal{C}[Q]) \notin \mathcal{R}$ then $\lambda\mathcal{C}[\,]$ would be a context for which the definition of Φ_d does not hold. So $(P,Q) \notin \Phi_d(\mathcal{R})$ and $\mathcal{R} \not\subseteq \Phi_d(\mathcal{R})$. ($\Leftarrow$) Obvious. □

As a corollary, we have that Dynamic Equivalence is a Congruence, i.e. $P \approx^d Q$ if and only if $\mathcal{C}[P] \approx^d \mathcal{C}[Q]$ for each context $\mathcal{C}[\,]$. Since $\approx^c$ is the coarsest congruence contained in $\approx$ and $\approx^d \subseteq \approx$, it follows that $\approx^d \subseteq \approx^c$.

Proposition 3.4 *(Dynamic Bisimulation $\Leftrightarrow$ Bisimulation and Congruence)*
$\mathcal{R} \in \mathcal{B} \cap \mathcal{C}$ if and only if $\mathcal{R} \subseteq \Phi_d(\mathcal{R})$.
Proof. ($\Rightarrow$) $\mathcal{R} \in \mathcal{B}$ implies $\mathcal{R} \subseteq \Phi(\mathcal{R})$ and $\mathcal{R} \in \mathcal{C}$ gives that $P\mathcal{R}Q$ implies $\mathcal{C}[P]\mathcal{R}\mathcal{C}[Q]$ $\forall \mathcal{C}[\,]$. Then if $(P,Q) \in \mathcal{R}$, $\forall \mathcal{C}[\,]$ $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$ and so $(\mathcal{C}[P], \mathcal{C}[Q]) \in \Phi(\mathcal{R})$ which, by definition of Φ, implies $(P,Q) \in \Phi_d(\mathcal{R})$. Therefore, $\mathcal{R} \subseteq \Phi_d(\mathcal{R})$. ($\Leftarrow$) If $R \subseteq \Phi_d(\mathcal{R})$ then $\mathcal{R} \subseteq \Phi(\mathcal{R})$ by lemma 3.2, so $\mathcal{R} \in \mathcal{B}$. Moreover if $(P,Q) \in \mathcal{R}$ then by lemma 3.3 $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$, so $\mathcal{R} \in \mathcal{C}$. □

Therefore, $\approx^d = \bigcup\{\mathcal{R} | \mathcal{R} \in \mathcal{B} \cap \mathcal{C}\}$ is the coarsest bisimulation which is also a congruence.

3.2 Algebraic characterization

In this subsection we show that $\approx^d$ has a corresponding object in **CatLCCS**, the category of CCS computations whose construction is due to Ferrari and Montanari ([FM90]).

As we have seen in section 2, the operational semantics of CCS is defined by a deductive system. Now, we structure those systems as *typed algebras* ([MSS90]), i.e. algebras in which types are assigned to elements, and which contain, as special elements, the types themselves.

Types allow us to distinguish between elements which are agents (typed by *state* and denoted by $u, v, w \ldots$), elements which are transitions (typed by $\rightarrow$ and denoted by t) and elements which are computations (typed by $\Rightarrow$ and denoted by c).

In the following $x : type$ will indicate that x has type *type*. Operations $source()$ and $target()$ and a function $label()$ which respectively give source state, target state and observation, are defined on elements typed by $\rightarrow$ or $\Rightarrow$. We write $t : u \stackrel{\mu}{\longrightarrow} v$ to denote a transition with $source(t) = u$, $target(t) = v$ and $label(t) = \mu$. Similarly, we write $c : u \stackrel{s}{\Longrightarrow} v$. A computation with empty observation will be indicated by $c : u \stackrel{\epsilon}{\Longrightarrow} v$, while we will write $u \Rightarrow v$ when we are not interested in the observation.

The definition of CCS models should be given by listing an appropriate *presentation* and saying that CCS models are the models of that presentation. Since such a presentation would be rather long, we prefer to give the definition as follows. The interested reader can find the rigorous definition in [FMM91].

Definition 3.5 *(CCS Models and Morphisms,* **CatLCCS***)*
A CCS Model is a typed algebra (with multityping) where elements typed state *have the algebraic structure of guarded CCS agents. Moreover, there is an operation on transitions for each rule in the CCS transition system, an operation idle and an operation* $\_;\_$*. They satisfy the following:*

$$[\mu, v >: \mu.v \xrightarrow{\mu} v$$

$$\frac{t : u \xrightarrow{\mu} v}{t[\Phi] : u[\Phi] \xrightarrow{\Phi(\mu)} v[\Phi]} \qquad \frac{t : u \xrightarrow{\mu} v}{t \setminus \alpha : u \setminus \alpha \xrightarrow{\mu} v \setminus \alpha} \text{ if } \mu \notin \{\alpha, \overline{\alpha}\}$$

$$\frac{t : u \xrightarrow{\mu} v}{t <+ w : u + w \xrightarrow{\mu} v} \qquad \frac{t : u \xrightarrow{\mu} v}{w +> t : w + u \xrightarrow{\mu} v}$$

$$\frac{t : u \xrightarrow{\mu} v}{t \rfloor w : u | w \xrightarrow{\mu} v | w} \qquad \frac{t : u \xrightarrow{\mu} v}{w \lfloor t : w | u \xrightarrow{\mu} w | v}$$

$$\frac{t_1 : u_1 \xrightarrow{\lambda} v_1 \text{ and } t_2 : u_2 \xrightarrow{\overline{\lambda}} v_2}{t_1 | t_2 : u_1 | u_2 \xrightarrow{\tau} v_1 | v_2}$$

$$\frac{t : u \xrightarrow{\lambda} v}{t : u \overset{\lambda}{\Longrightarrow} v} \qquad \frac{t : u \xrightarrow{\tau} v}{t : u \overset{\epsilon}{\Longrightarrow} v} \qquad idle(v) : v \overset{\epsilon}{\Longrightarrow} v \qquad \frac{c_1 : u \overset{s_1}{\Longrightarrow} v \text{ and } c_2 : v \overset{s_2}{\Longrightarrow} w}{c_1; c_2 : u \overset{s_1 s_2}{\Longrightarrow} w}$$

Finally, a CCS Model satisfies the following equations:

$$rec x.u = u[rec x.u/x] \qquad c_1; (c_2; c_3) = (c_1; c_2); c_3 \qquad \frac{c : u \Rightarrow v}{idle(u); c = c = c; idle(v)}$$

A CCS morphism is an homomorphism of algebras that respects types. This defines **CatLCCS***, the category whose objects are CCS Models and whose morphisms are CCS morphisms.* □

Note that the way in which we defined the operations also defines *source*, *target* and *label*. Note also that there are no rules and operations for recursion which is instead handled by imposing the axiom above on states. Moreover, τ's are completely forgotten in the observations. Finally, note that a CCS morphism respects *source* and *target* since they are operations of the algebra. It is easy to prove that CCS morphisms respect the function *label*.

As a general result on typed algebras ([MSS90]), we state that **CatLCCS** has an initial object $\Im$.

Weak Observational Congruence cannot be characterized algebraically in **CatLCCS**, even though Ferrari and Montanari showed ([FM90]) that this is possible in a category constructed ad hoc from it. This is because the definition of morphism implies that, from congruent states, corresponding transitions lead to congruent states, and this is *not* the case for Weak Observational Congruence.

The situation is different for $\approx^d$. In the following, we shall use $[P]$ to denote the state to which agent P evaluates in a particular CCS model.

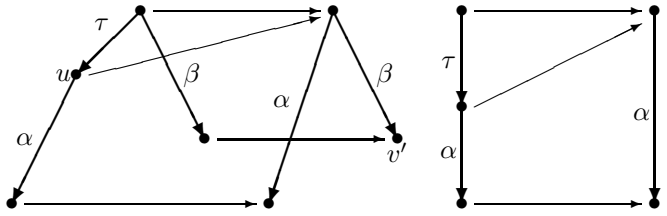
The following lemma derives directly from the fact that h respects the algebraic structure of elements.

Lemma 3.6 *(CatLCCS morphisms respect contexts)*
If h is a CatLCCS morphism then $h([P]) = h([Q])$ implies $h([\mathcal{C}[P]]) = h([\mathcal{C}[Q]])$ for each context $\mathcal{C}[\ ]$. □

Definition 3.7 *(Transition Preserving Homomorphism)*
A CatLCCS morphism $h : C \to C'$ is called a transition preserving homomorphism if and only if:

- *$h : C \to C'$ is a surjective CCS morphism;*
- *$t' \in C',\ t' : h(u) \Rightarrow v'$ implies $\exists t \in C,\ t : u \Rightarrow v$ with $h(t) = t'$.* □

Example 3.8 *(no tp-homomorphism maps $\tau.\alpha + \beta$ to $\alpha + \beta$ or $\tau.\alpha$ to α)*



(A) **(B)**

The figures show two morphisms which are not tp-morphisms.
In case (A) we have $t : h(u) \overset{\beta}{\Longrightarrow} v'$ but $u \overset{\beta}{\not\Longrightarrow}$.
In case (B) the morphism cannot respect the algebra, for if it did we would have $h(\tau.\alpha) + h(\beta) = h(\alpha) + h(\beta)$ and so $h(\tau.\alpha + \beta) = h(\alpha + \beta)$ which is case (A). □

Proposition 3.9 *(tp-homomorphism ⇒ Dynamic Bisimulation)*
If $h : \Im \to C$ is a tp-homomorphism then $h([P]) = h([Q])$ implies $P \approx^d Q$.

Proof. We show that $\mathcal{R} = \{< P, Q > | h([P]) = h([Q])\}$ is a dynamic bisimulation. Let $(P, Q) \in \mathcal{R}$. If $\mathcal{C}[P] \stackrel{s}{\Longrightarrow} P'$ then there exists $t : [\mathcal{C}[P]] \stackrel{s}{\Longrightarrow} [P']$ in $\Im$. Then $h(t) : h([\mathcal{C}[P]]) \stackrel{s}{\Longrightarrow} h([P'])$. By lemma 3.6 we have $h(t) = t' : h([\mathcal{C}[Q]]) \stackrel{s}{\Longrightarrow} h([P'])$ and so by definition of tp-homomorphism there exists $t'' : [\mathcal{C}[Q]] \stackrel{s}{\Longrightarrow} [Q']$ with $h([Q']) = h([P'])$. Hence there exists $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$. Symmetrically if $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$. Therefore, by definition of Φ_d, $\mathcal{R} \subseteq \Phi_d(\mathcal{R})$. □

Proposition 3.10 *(Dynamic Congruence ⇒ tp-homomorphism)*
There exists an object $\Im/\approx^d$ of **CatLCCS** *such that the unique morphism $h_{\approx^d} : \Im \to \Im/\approx^d$ is a tp-homomorphism. Moreover, $P \approx^d Q$ implies $h_{\approx^d}([P]) = h_{\approx^d}([Q])$.*

Proof. Let $\mathcal{R}$ be the congruence over $\Im$ defined as follows:
$[P]\mathcal{R}[Q]$ iff $P \approx^d Q$ and $(t_1 : u_1 \stackrel{s}{\Longrightarrow} v_1)\mathcal{R}(t_2 : u_2 \stackrel{s}{\Longrightarrow} v_2)$ iff $u_1\mathcal{R}u_2$ and $v_1\mathcal{R}v_2$.
$\Im/\approx^d$ is obtained as the quotient of $\Im$ modulo $\mathcal{R}$ and $h_{\approx^d} : \Im \to \Im/\approx^d$ is the canonical map which sends each element to its equivalence class. □

Hence, as a corollary of the previous two propositions we have that $h_{\approx^d}$ coincides with $\approx^d$, i.e. $P \approx^d Q$ if and only if $h_{\approx^d}([P]) = h_{\approx^d}([Q])$.

Moreover, $\Im/\approx^d$ is the terminal object in the subcategory of the objects reachable from $\Im$ through tp-homomorphisms, that is the one corresponding to the coarsest dynamic bisimulation, i.e. $\approx^d$.

Proposition 3.11 *($\Im/\approx^d$ is terminal)*
The subcategory of **CatLCCS** *consisting of all objects reachable from $\Im$ through tp-homomorphisms and having morphisms which are tp-homomorphisms has $\Im/\approx^d$ as a terminal object.* □

4 Progressing Bisimulation

In this section we introduce a new bisimulation, *Progressing Bisimulation*, on the states of a labelled transition system with a distinct label τ. We will give an *algebraic characterization* of such a bisimulation and two *modal logical languages*, in the style of HML, adequate with respect to it. Furthermore we will provide a *complete axiomatization* of Progressing Equivalence for states with finite computations.

In the next section we will see that, when the transition system is the CCS transition system, Progressing Bisimulation coincides with Dynamic Congruence, thus completing its characterization for CCS.

We reiterate our two distinct results: the first, concerning Dynamic Congruence and guided by the concept of context, and the second concerning Progressing Bisimulation and its algebraic, logical and axiomatic characterizations. Both bisimulations are very general and go beyond CCS semantics, even though Dynamic Congruence is perhaps better justified in terms of practical considerations. Moreover, in the case of CCS they coincide, giving us plenty of characterizations of Dynamic Congruence.

4.1 Operational definition and Algebraic characterization

Definition 4.1 *(Progressing Bisimulation)*
Let $\mathcal{R}$ be a binary relation over the states of a labelled transition system $T =< S, \longrightarrow, \Lambda \cup \{\tau\} >$.
Then Φ_p, a function from relations to relations, is defined as follows:

$(s, r) \in \Phi_p(\mathcal{R})$ *if and only if* $\forall \mu \in \Lambda \cup \{\tau\}$*:*

- *whenever $s \stackrel{\mu}{\longrightarrow} s'$ there exists r' s.t. $r \stackrel{\mu}{\Longrightarrow} r'$ and $(s', r') \in \mathcal{R}$;*
- *whenever $r \stackrel{\mu}{\longrightarrow} r'$ there exists s' s.t. $s \stackrel{\mu}{\Longrightarrow} s'$ and $(s', r') \in \mathcal{R}$.*

A relation $\mathcal{R}$ is called progressing bisimulation*, if and only if $\mathcal{R} \subseteq \Phi_p(\mathcal{R})$.*
The relation $\approx^p = \cup\{\mathcal{R} | \mathcal{R} \subseteq \Phi_p(\mathcal{R})\}$ is called Progressing Equivalence*.* □

Relation $\approx^p$ is a progressing bisimulation and an equivalence relation.

Now we introduce an algebraic model of a labelled transition system. As for CCS Models (definition 3.5) the definition of LTS Models could be given more formally. The notations used here are those defined in the previous section.

Definition 4.2 *(LTS Models and Morphisms,* **LTS***)*
An LTS Model is a typed algebra (with multityping) where elements typed state *are a set, i.e. they do not have any particular algebraic structure. Partial operations idle and* $\_;\_$ *are defined so that they satisfy:*

$$\frac{t : u \xrightarrow{\lambda} v}{t : u \overset{\lambda}{\Longrightarrow} v} \qquad \frac{t : u \xrightarrow{\tau} v}{t : u \overset{\epsilon}{\Longrightarrow} v} \qquad idle(v) : v \overset{\epsilon}{\Longrightarrow} v \qquad \frac{c_1 : u \overset{s_1}{\Longrightarrow} v \text{ and } c_2 : v \overset{s_2}{\Longrightarrow} w}{c_1; c_2 : u \overset{s_1 s_2}{\Longrightarrow} w}$$

Moreover, an LTS Model satisfies the following equations:

$$c_1; (c_2; c_3) = (c_1; c_2); c_3 \qquad \frac{c : u \Rightarrow v}{idle(u); c = c = c; idle(v)}$$

An LTS morphism is a morphism of algebras that respects types and labelling. This defines **LTS***, the category whose objects are LTS Models and whose morphisms are LTS morphisms.* □

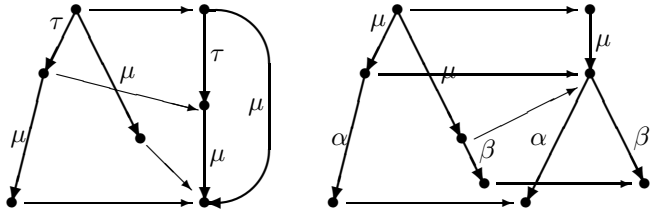
Clearly, given an LTS Model, elements typed by $\rightarrow$ represent transitions, elements typed by $\Rightarrow$ represent sequences of transitions (computations) and elements typed by *state* represent states of the transition system. Note that an LTS morphism also respects *source* and *target*.

We introduce a new kind of morphism which, besides preserving transitions, prevents τ-transitions to be mapped to *idle*-transitions. We refer to them as *progressing transition preserving morphisms*.

Definition 4.3 *(Progressing Transition Preserving Morphism)*
An LTS morphism $h : T \rightarrow T'$ is called a progressing transition preserving morphism if and only if:

- $h : T \rightarrow T'$ *is a surjective LTS morphism;*
- $t' \in T',\ t' : h(s) \Rightarrow r'$ *implies* $\exists t \in T,\ t : s \Rightarrow r$ *with* $h(t) = t'$*;*
- $h(t) = idle(h(s))$ *implies* $t = idle(s)$. □

Example 4.4 *(ptp-morphisms map $\tau.\mu + \mu$ to $\tau.\mu$ but not $\mu.\alpha + \mu.\beta$ to $\mu.(\alpha + \beta)$)*



(C) **(D)**

Cases (A) and (B) of example 3.8 are not ptp-morphisms, the first because it does not preserve transitions, the second because it maps a not–idle to an idle transition.
In case (C) we have a ptp-morphism mapping a $\xrightarrow{\mu}$ transition to a computation $\xrightarrow{\tau}; \xrightarrow{\mu} = \overset{\mu}{\Longrightarrow}$, while the morphism in case (D) is not a ptp, for it does not preserve transitions. □

The following proposition establishes the correspondence between ptp-morphisms and progressing bisimulations. This result is very similar to that in section 3.2.

Proposition 4.5 *(ptp-morphism ⇔ Progressing Bisimulation)*
If $h : T \rightarrow T'$ is a ptp-morphism then $h(s) = h(r)$ implies $s \approx^p r$.
There exist $T/\approx^p$ and a ptp-morphism $h_{\approx^p} : T \rightarrow T/\approx^p$ such that $s \approx^p r$ implies $h_{\approx^p}(s) = h_{\approx^p}(r)$.
Therefore, $s \approx^p r$ if and only if $h_{\approx^p}(s) = h_{\approx^p}(r)$.
Finally, the subcategory of **LTS** *consisting of all objects reachable from T through ptp-morphisms, and having morphisms which are ptp-morphisms, has $T/\approx^p$ as a terminal object.* □

4.2 Logical characterization

In this subsection we design two modal logical languages in the style of HML which are adequate with respect to Progressing Bisimulation, i.e. two states are progressing equivalent if and only if they cannot be distinguished by any formula of the language.

The proofs in the rest of the section follow Milner's scheme in [Mil89], and so they are sketched very roughly. For the definitions of the modal languages adequate with respect to Strong Congruence and Weak Equivalence see [Mil89, chapter 10].

We introduce now a language whose modal operator may consider τ's, with the meaning that at least those τ's must be observed in the models. In the following, A^+ will mean $A^* \setminus \{\epsilon\}$.

Definition 4.6 *(The language : $PL_\tau^{\approx^p}$)*
$PL_\tau^{\approx^p}$ is the smallest class of formulae containing the following:

- *If $\varphi \in PL_\tau^{\approx^p}$ then $\forall t \in (\Lambda \cup \{\tau\})^+$ $\langle\langle t \rangle\rangle_\tau \varphi \in PL_\tau^{\approx^p}$*
- *If $\varphi \in PL_\tau^{\approx^p}$ then $\neg\varphi \in PL_\tau^{\approx^p}$*
- *If $\varphi_i \in PL_\tau^{\approx^p}$ $\forall i \in I$ then $\bigwedge_{i\in I} \varphi_i \in PL_\tau^{\approx^p}$, where I is an index set.* □

Definition 4.7 *(Satisfaction relation)*
The validity of a formula $\varphi \in PL_\tau^{\approx^p}$ at state r ($r \models_\tau \varphi$) is inductively defined as follows :

- *$r \models_\tau \langle\langle t \rangle\rangle_\tau \varphi$ if and only if $\exists r'$ s.t. $r \stackrel{t}{\Longrightarrow} r'$ and $r' \models_\tau \varphi$*
- *$r \models_\tau \neg\varphi$ if and only if* not *$r \models_\tau \varphi$*
- *$r \models_\tau \bigwedge_{i\in I} \varphi_i$ if and only if $\forall i \in I$ $r \models_\tau \varphi_i$.* □

There is another modal language we can naturally associate to $\approx^p$: $PL^{\approx^p}$. Its syntax is obtained from that of $PL_\tau^{\approx^p}$ by substituting the modal operator $\langle\langle t \rangle\rangle_\tau$ with the operator $\langle\langle s \rangle\rangle_p$ for $s \in \Lambda^*$.
The satisfaction relation $\models$ is obtained by replacing in definition 4.7 the clause for $r \models_\tau \langle\langle t \rangle\rangle_\tau \varphi$ with the clause $r \models \langle\langle s \rangle\rangle_p \varphi$ if and only if $\exists r'$ s.t. $r \stackrel{\overline{s}}{\Longrightarrow} r'$ and $r' \models \varphi$, where $\overline{s} =$ (if $s \neq \epsilon$ then s else τ).

The difference between the two languages is that $PL^{\approx^p}$ does not consider τ's in its modal operator, but takes care of them in the satisfaction relation, while the reverse holds for $PL_\tau^{\approx^p}$. The language used is a matter of taste.

Proposition 4.8 *($PL^{\approx^p}$ and $PL_\tau^{\approx^p}$ induce the same equivalence)*
$\forall \psi \in PL^{\approx^p}$ $s \models \psi \Leftrightarrow r \models \psi$ if and only if $\forall \varphi \in PL_\tau^{\approx^p}$ $s \models_\tau \varphi \Leftrightarrow r \models_\tau \varphi$. □

We show that the equivalences induced by $PL^{\approx^p}$ and $PL_\tau^{\approx^p}$ coincide with $\approx^p$.

Proposition 4.9 *($PL_\tau^{\approx^p}$ is adequate w.r.t. $\approx^p$)*
$s \approx^p r$ if and only if $\forall \varphi \in PL_\tau^{\approx^p}$ $s \models_\tau \varphi \Leftrightarrow r \models_\tau \varphi$.

Proof. Following Milner's scheme, we define stratifications $PL_{\tau\kappa}^{\approx^p}$ and $\approx_\kappa^p$, respectively of $PL_\tau^{\approx^p}$ and $\approx^p$, for κ an ordinal, and prove that for each κ, $s \approx_\kappa^p r$ if and only if $\forall \varphi \in PL_{\tau\kappa}^{\approx^p}$ $s \models_\tau \varphi \Leftrightarrow r \models_\tau \varphi$. The proposition follows easily from that. □

As a corollary to propositions 4.8 and 4.9, we have that also $PL^{\approx^p}$ is adequate w.r.t. $\approx^p$, that is $s \approx^p r$ if and only if $\forall \psi \in PL^{\approx^p}$ $s \models \psi \Leftrightarrow r \models \psi$.

4.3 Axiomatic characterization

Going back to CCS, in this subsection we give a complete axiomatization of $\approx^p$ for finite CCS agents. It is worth noticing that every labelled transition system with finite computations can be represented by a finite sequential CCS agent, in a straightforward way.

Obviously, to carry on a proof with axioms and equational deduction, we need an observational equivalence which is actually a congruence. For the moment let us assume that $\approx^p$ is a congruence with respect the CCS operators: in the next section, we will prove that this is indeed the case (see proposition 5.1).
For the axiomatizations of Strong and Weak Observational Congruence see [Mil89, pp. 160–165].
Let us begin relating $\approx^p$ on CCS agents to $\sim$ and $\approx^c$. The following is a direct consequence of the definitions of Ψ and Φ_p.

Proposition 4.10 *(Strong Congruence refines Progressing Bisimulation)*
$\sim \subseteq \approx^p$, where $\sim$ is Strong Observational Congruence. □

Thus $\approx^p$ inherits all the properties of $\sim$, in particular *monoid laws* and the *expansion theorem* ([Mil89, pp. 62, 67–76]) hold for $\approx^p$. Concerning the τ-laws ([Mil89, p. 62]) we have:

Proposition 4.11 *(Progressing Bisimulation and τ-laws)*
i. $P + \tau.P \approx^p \tau.P$ *ii.* $\alpha.(P + \tau.Q) + \alpha.Q \approx^p \alpha.(P + \tau.Q)$ *iii.* $\alpha.\tau.P \not\approx^p \alpha.P$ □

A CCS agent is *finite* if it does not contain recursion, and it is *serial* if it contains no parallel composition, restriction or relabelling. It is clear that with the use of the *expansion theorem* every finite agent can be equated to a finite serial agent. Therefore, a complete axiomatization for finite and serial agents can be considered an axiomatization for finite agents (considering the expansion theorem as an axiom scheme). In the rest of the subsection every CCS agent must be considered finite and serial.

We introduce a new set of axioms similar to the ones given for Strong and Weak Observational Congruence: it contains the monoid laws and two of the three τ-laws.

Definition 4.12 *(Axioms System $\mathcal{A}$)*

$$\begin{array}{ll} \mathbf{A_1} : P+Q=Q+P & (\mathbf{T_1} : \mu.\tau.P=\mu.P) \\ \mathbf{A_2} : P+(Q+R)=(P+Q)+R & \mathbf{T_2} : P+\tau.P=\tau.P \\ \mathbf{A_3} : P+P=P & \mathbf{T_3} : \mu.(P+\tau.Q)+\mu.Q=\mu.(P+\tau.Q) \\ \mathbf{A_4} : P+nil=P & \end{array}$$

$$\mathcal{A}=\{\mathbf{A_1},\mathbf{A_2},\mathbf{A_3},\mathbf{A_4}\}\cup\{\mathbf{T_2},\mathbf{T_3}\}$$

□

Now, we prove that $\mathcal{A}$ is a complete set of axioms for $\approx^p$, i.e. two agents are progressing equivalent if and only if they can be proved equal by the axioms of $\mathcal{A}$ and equational deduction (denoted by $\vdash$).

Proposition 4.13 *($\mathcal{A}$ is a complete axiomatization of $\approx^p$)*
$P \approx^p Q$ if and only if $\mathcal{A} \vdash P=Q$.

Proof. ($\Leftarrow$) As previously noticed axioms $\mathcal{A}$ are true for $\approx^p$.
($\Rightarrow$) Following Milner's scheme, we define a standard form (SF) for CCS agents such that, using axioms $\mathbf{A_3}$, $\mathbf{A_4}$, $\mathbf{T_2}$ and $\mathbf{T_3}$, we can prove that each P is equal to a P' in SF with $P \approx^p P'$. By induction on the number of nested prefixes, it is easy to show that, if $P \approx^p Q$ and if P and Q are in SF, they can be proved equal using axioms $\mathbf{A_1}$ and $\mathbf{A_2}$. So, if $P \approx^p Q$ then $\mathcal{A} \vdash P=Q$. □

5 Dynamic Congruence and Progressing Bisimulation

In this section we show that Dynamic Congruence and Progressing Bisimulation *coincide* when $\approx^p$ is considered on CCS.

This gives many characterizations to $\approx^d$: in fact, we have two characterizations by finality through particular kinds of *abstraction morphisms* (the one encoding the CCS algebra into states and transitions, the other just considering the naked labelled transition system), two logical characterizations via HML-like modal languages and, finally, an axiomatization for finite agents, besides the two operational characterizations given by the *bisimulation game*.

Proposition 5.1 *($\approx^p$ is a congruence)*
Let E, F be CCS expressions and $E \approx^p F$. Then

$$\begin{array}{lll} i..\ \mu.E \approx^p \mu.F & ii..\ E+Q \approx^p F+Q & iii..\ E|Q \approx^p F|Q \\ iv..\ E[\Phi] \approx^p F[\Phi] & v..\ E\setminus\alpha \approx^p F\setminus\alpha & vi..\ recx.E \approx^p recx.F \end{array}$$

Proof. The proof is standard. For *i–v* it is enough to exhibit appropriate progressing bisimulations. For instance, $\mathcal{R}=\{(P_1|Q, P_2|Q) | P_1 \approx^p P_2\}$ shows *iii*. Point *vi* is less trivial and must be done by induction on the depth of the proofs by which actions are inferred, exploiting a concept analogous to Milner's bisimulation up to $\approx^c$. □

Proposition 5.2 *(Dynamic and Progressing Bisimulations coincide)*
$\approx^d = \approx^p$.

Proof. We shall prove that $\approx^d \subseteq \approx^p$ showing that $\approx^d \subseteq \Phi_p(\approx^d)$. Suppose $P \approx^d Q$ and $P \xrightarrow{\mu} P'$. If $\mu \neq \tau$ then we have that $\exists Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \approx^d Q'$. Otherwise, if $\mu=\tau$ it must be that $\exists Q \stackrel{\epsilon}{\Longrightarrow} Q'$ in which at least one τ move is done and $P' \approx^d Q'$. Actually, if this were not the case, we could find a context for which the definition of $\approx^d$ does not hold: $\mathcal{C}[\ ] \equiv x+\alpha.\overline{P}$, where $\overline{P}$ is not dynamically equivalent to each α-derivate of P', if there exists any, otherwise $\overline{P} \equiv nil$ will do.

Symmetrically if $Q \xrightarrow{\mu} Q'$. Then we have $(P,Q) \in \Phi_p(\approx^d)$ and so $\approx^d \subseteq \Phi_p(\approx^d)$.
By similar technique we show that $\approx^p \subseteq \approx^d$. Suppose that $P \approx^p Q$. Since $\approx^p$ is a congruence then $\forall \mathcal{C}[\ ]\ \mathcal{C}[P] \approx^p \mathcal{C}[Q]$. If $\mathcal{C}[P] \stackrel{s}{\Longrightarrow} P'$ with $s \in \Lambda^*$ then, by repeated application of the definition of $\approx^p$, we have $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$ and $P' \approx^p Q'$. Symmetrically if $\mathcal{C}[Q] \stackrel{s}{\Longrightarrow} Q'$. So $\approx^p \subseteq \Phi_d(\approx^p)$. □

Acknowledgements. We wish to thank GianLuigi Ferrari for many interesting discussions, which have significantly contributed to our understanding of the subject. Special thanks to Rocco De Nicola for his suggestions on the organization of the paper.

References

[**Abr88**] S. Abramsky. *A Domain Equation for Bisimulation*. Technical report, Department of Computing, Imperial College, London, 1988.

[**Acz87**] P. Aczel. *Non-Well-Founded Sets*. CSLI Lecture Notes n. 14, Stanford University, 1987.

[**BBS88**] D. Benson and O. Ben-Shachar. Bisimulations of Automata. *Information and Computation*, n. 79, 1988.

[**DDM90**] P. Degano, R. De Nicola, and U. Montanari. A Partial Ordering Semantics for CCS. *Theoretical Computer Science*, n. 75, 1990.

[**De 87**] R. De Nicola. Extensional Equivalence for Transition Systems. *Acta Informatica*, n. 24, 1987.

[**FM90**] G. Ferrari and U. Montanari. Towards the Unification of Models for Concurrency. In *Proceedings of CAAP '90*. LNCS n. 431, Springer Verlag, 1990.

[**FMM91**] G. Ferrari, U. Montanari, and M. Mowbray. On Causality Observed Incrementally, Finally. In *Proceedings of TAPSOFT '91*. LNCS n. 493, Springer Verlag, 1991.

[**GvG89**] U. Goltz and R. van Glabbeek. Equivalence Notions for Concurrent System and Refinement of Actions. In *Proceedings of MFCS '89*. LNCS n. 379, Springer Verlag, 1989.

[**Kel76**] R. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, vol. 7, 1976.

[**Mil80**] R. Milner. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, n. 92. Springer Verlag, 1980.

[**Mil89**] R. Milner. *Concurrency and Communication*. Prentice Hall, 1989.

[**MS89**] U. Montanari and M. Sgamma. Canonical Representatives for Observational Equivalences Classes. In *Proceedings of Colloquium on the Resolution of Equations in Algebraic Structures*. North Holland, 1989.

[**MS90**] U. Montanari and V. Sassone. *Dynamic Bisimulation*. Technical Report TR 13/90, Dipartimento di Informatica, Università di Pisa, 1990.

[**MSS90**] V. Manca, A. Salibra, and G. Scollo. Equational Type Logic. *Theoretical Computer Science*, n. 77, 1990.

[**NPW81**] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, n. 13, 1981.

[**Par81**] D. Park. Concurrency and Automata on Infinite Sequences. In *Proceedings of GI*. LNCS n. 104, Springer Verlag, 1981.

[**Pet62**] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[**Plo81**] G. Plotkin. *A Structured Approach to Operational Semantics*. DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.

[**Pra86**] V. Pratt. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, n. 15, 1986.

[vG87] R. van Glabbeek. Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra. In *Proceedings of STACS 87*. LNCS n. 247, Springer Verlag, 1987.

[vGW89] R. van Glabbeek and W. Weijland. Branching Time and Abstraction in Bisimulation Semantics. In *Proceedings of IFIP 11th World Computer Congress*, August 1989.