Performance Preorder: Ordering Processes with respect to Speed *

Flavio Corradini[1] and Roberto Gorrieri[2] and Marco Roccetti[2]

[1] Dipartimento di Scienze dell'Informazione, Università "La Sapienza"
Via Salaria 113, I-00189 Roma, Italy
e-mail: corradini@dsi.uniroma1.it

[2] Dipartimento di Scienze dell'Informazione, Università di Bologna
Piazza di Porta S. Donato 5, I-40127 Bologna, Italy
e-mail: {gorrieri,roccetti}@cs.unibo.it

Abstract. The theory of processes with durational actions proposed in [8] is equipped here with a preorder based on execution speed, called performance preorder. Two processes P and Q are related if they are strong bisimilar (i.e., functional equivalent) and the first one is at least as fast as the second one. Hence, this preorder supports the stepwise refinement "from specification to implementation" by increasing efficiency while retaining the same functionality. We show that the problem of finding faster implementations for a specification is connected to the problem of finding more distributed implementations of the same specification. This is an immediate consequence of the proof that the location preorder, which is based on a measure of distribution, implies the performance preorder.

1 Introduction

Recently, in the field of semantics for process description languages, a great deal of interest has been stirred up by equivalence notions which incorporate some measure of efficiency (see, e.g., [12, 3, 8]). Some of them [2, 7, 8] rely on the basic idea that processes are discriminated not only according to their *functionality* (what actions the processes can do), but also considering their *performance* (a measure of the time consumed for their execution).

Any action a is assumed to have a duration $f(a)$ which represents the time needed for its execution. Every sequential subprocess is equipped with a clock, whose elapsing is set by the execution of actions. Whenever an action a is executed by a sequential subcomponent F, the value n of the clock of F is incremented to $n + f(a)$, whilst the local clocks of those sequential components not involved in the execution of a are unaffected. Hence, if F is idle during a transition, its local clock value cannot increase. In other words, each sequential subprocess is always *eager* to perform an executable action (or dually actions are *urgent*): the time value is incremented locally only when the executable action is performed. A simple example may be helpful. Consider the term $E = a.c|b$. Since

* Research partially supported by MURST, CNR and Esprit BRA n. 8130 LOMAPS.

the clock is set to 0 before starting the execution of E, $s = (0 \Rightarrow a.c) \mid (0 \Rightarrow b)$ is the initial state of the transition system, where the auxiliary operator $n \Rightarrow F$ means that the execution of F starts exactly after n time units of the global clock. One of the two transitions out of s is labelled $< a, f(a) >$ and reaches $(f(a) \Rightarrow c)|(0 \Rightarrow b)$. By executing b, it reaches the state $(f(a) \Rightarrow c)|(f(b) \Rightarrow nil)$; finally the execution of c produces a transition labelled $< c, f(a) + f(c) >$ with target state $(f(a) + f(c) \Rightarrow nil) \mid (f(b) \Rightarrow nil)$. It is immediate observing that the time needed for the complete execution of E is $max\{f(a)+f(c), f(b)\}$. Standard bisimulation on this labelled transition system defines a new semantics for process algebras, called *performance equivalence* in [8], which equates systems whenever they perform the *same* actions in the *same* amount of time.

This paper is in the line of [8], but here we focus on the problem of *ordering processes with respect to speed*, allowing to consider degrees of efficiency among processes, for example by forcing implementations to be faster than (although functionally equivalent to) specifications. The tool used to this aim is a preorder, called *performance preorder* and denoted by $\sqsubseteq_p$, which is based on the classical notion of bisimulation. Intuitively, $E_1 \sqsubseteq_p E_2$ if and only if E_1 and E_2 are strong bisimilar but E_1 is at least as fast as E_2. The kernel of this preorder gives rise to a new equivalence, called *competitive equivalence* and denoted $\simeq_c$, which relates processes by considering only the optimal (or fastest) and worst (or slowest) executions. This equivalence is more discriminating than interleaving bisimulation [10]. Indeed, the equation $a \mid b \simeq_c a.b + b.a$ does not hold in our setting because $a \mid b \sqsubseteq_p a.b+b.a$ but $a.b+b.a \not\sqsubseteq_p a \mid b$. To see this consider the possible transitions which $a.b + b.a$ and $a \mid b$ can perform:

$$0 \Rightarrow a.b + b.a \stackrel{<a,f(a)>}{\longrightarrow} f(a) \Rightarrow b \stackrel{<b,f(a)+f(b)>}{\longrightarrow} f(a) + f(b) \Rightarrow nil$$

$$0 \Rightarrow a \mid 0 \Rightarrow b \stackrel{<a,f(a)>}{\longrightarrow} f(a) \Rightarrow nil \mid 0 \Rightarrow b \stackrel{<b,f(b)>}{\longrightarrow} f(a) \Rightarrow nil \mid f(b) \Rightarrow nil$$

Thus, we can see that all the actions of the former state can be performed in a less or equal time of the corresponding "matching" transitions of the latter state. Competitive equivalence is weaker than performance equivalence: $E_1 = a.a.a + a|a|a$ and $E_2 = a.a.a + a|a|a + a.a|a$ provide the needed example. We claim that competitive equivalence is more appropriate when one is interested in extimating the time intervals for the optimal and worst cases only.

The paper is organized as follows. In Section 2 we introduce a simple process algebra (with recursion, but without communication), together with its operational semantics. In Section 3 we recall performance equivalence from [8] and describe performance preorder and competitive equivalence. In Section 4 we compare competitive equivalence with the best known untimed non-interleaving semantics proposed in the literature. This includes a comparison of performance preorder with the *causal preorder* [1] and with the *location preorder* [4]. The latter comparison is particularly interesting as it proves the (quite expected) result that the more distributed a process is, the more efficient it is. Finally, Section 5 is devoted to comparison with related work and further research.

2 Language and Operational Semantics

Let Act be the set of *actions*, ranged over by a, b, c, ... and X be the set of variables, ranged over by x, y, z, ... The process terms are freely generated by the following syntax:

$$F \quad ::= \quad nil \;\Big|\; a.F \;\Big|\; F + F \;\Big|\; F \,|\, F \;\Big|\; x \;\Big|\; \mathrm{rec}\; x.\, F$$

This is the subset of *CCS* [10], where the parallel operator $|$ does not allow communication. In the following, we will focus on the set P of the closed terms (called *agents* or *processes*), ranged over by E. The set of *finite processes* (i.e., without recursion) will be denoted by P_{fin}.

The final goal of any program is to be executed, and the measure of the time it consumes strictly depends on the chosen machine. Our assumptions on abstract machines are the following:

1) *maximal parallelism:* whenever a new subprocess is activated, there is always a processor free, ready to execute it.
2) *eagerness:* there is no time-passing in between the execution of actions from the same subprocess; equivalently actions happen as soon as possible.
3) *static duration:* the amount of time needed for the execution of an action is fixed on the basis of the features of the abstract machine.

As the duration of actions can be different for different machines, we should take this parameter into account. One possibility is to introduce *duration functions* (ranged over by f, g) which associate to each action the number of time units needed for its execution. In the following, we choose any duration function $f : Act \rightarrow N^+$, simply to fix this parameter. Our language is equipped with an SOS semantics in terms of labelled transition systems.

Definition 1. A *labelled transition system* is a triple $< S, M, T >$ where S is a set of *states*, M is a set of *labels* and $T = \{ \stackrel{\mu}{\longrightarrow} \subseteq S \times S \mid \mu \in M \}$ is the *transition relation*. We will write $s \stackrel{\mu}{\longrightarrow} s'$ instead of $< s, s' > \in \stackrel{\mu}{\longrightarrow}$.

In our case, the states are terms of a syntax extending the one for agents with a local *clock prefixing* operator, $n \Rightarrow \_$, which records the evolution of different parts of a distributed state.

Definition 2. The states are terms generated by the following syntax:

$$s \quad ::= \quad n \Rightarrow nil \;\Big|\; n \Rightarrow a.E \;\Big|\; s + s \;\Big|\; s \,|\, s \;\Big|\; n \Rightarrow \mathrm{rec}\; x.\, F$$

where E and $\mathrm{rec}\; x.\, F$ denote agents. The set of states is denoted by S.

The equations in Table 1, called *clock distribution equations*, show that the operator $n \Rightarrow \_$ may be applied to any agent E, which can be canonically reduced to a state, when interpreting these equations as rewrite rules from left to right.

$$n \Rightarrow (E \mid E') \;=\; (n \Rightarrow E) \mid (n \Rightarrow E')$$
$$n \Rightarrow (E + E') = (n \Rightarrow E) + (n \Rightarrow E')$$

Table 1. Clock Distribution Equations.

Each transition is labelled by a pair of the form $< a, n >$ meaning that action a has been completed exactly n time units after the computation began. Hence, the set of labels for the transition system is $\Pi = Act \times N^{+}$. The transition relation is given through a set of inference rules (Table 2) defined in a structural inductive manner. It is worthwhile observing that these rules are parametric w.r.t. the chosen duration function f. Hence, to be precise, we should write $\rightarrow_f$.

$$Act \;\; \frac{}{n \Rightarrow a.E \xrightarrow{<a,k>} k \Rightarrow E} \qquad k = n + f(a)$$

$$Alt_1 \;\; \frac{s_1 \xrightarrow{<a,n>} s_1'}{s_1 + s_2 \xrightarrow{<a,n>} s_1'} \qquad Alt_2 \;\; \frac{s_2 \xrightarrow{<a,n>} s_2'}{s_1 + s_2 \xrightarrow{<a,n>} s_2'}$$

$$Par_1 \;\; \frac{s_1 \xrightarrow{<a,n>} s_1'}{s_1 \mid s_2 \xrightarrow{<a,n>} s_1' \mid s_2} \qquad Par_2 \;\; \frac{s_2 \xrightarrow{<a,n>} s_2'}{s_1 \mid s_2 \xrightarrow{<a,n>} s_1 \mid s_2'}$$

$$Rec \;\; \frac{n \Rightarrow E[\mathrm{rec}\ x.\ E/x] \xrightarrow{<a,n>} s'}{n \Rightarrow \mathrm{rec}\ x.\ E \xrightarrow{<a,n>} s'}$$

Table 2. The Structural Rules for the Operational Semantics.

A few comments are now in order. Rule Act states that an action a, executable at time n, is completed at time $k = n + f(a)$; the number k denotes the time which passed for all the sequential subprocesses of E. Note that $k \Rightarrow E$ may be not a state; in such a case, applications of the clock distribution equations will eventually transform $k \Rightarrow E$ into a state. Rules Alt_1, Alt_2, Par_1 and Par_2 are as usual. Finally, note that $n \Rightarrow E[\mathrm{rec}\ x.\ E/x]$ in the premise of rule Rec may need applications of the clock distribution equations to become a state.

3 Behavioural Semantics

In this section we first recall the definition of performance equivalence from [8] and prove that it is a congruence. Then, we study performance preorder and the induced competitive equivalence.

3.1 Performance Equivalence

The observational semantics we are interested in is based on the branching-time semantics of bisimulation; as a matter of fact, performance bisimulation is nothing but (ordinary) strong bisimulation on the transition system labelled by pairs of the form $< a, n >$.

Definition 3. (*Performance Equivalence*)

1) Let Rel denote the set of binary relations over S. The functional $R : Rel \rightarrow Rel$ is defined, for each $\Re \in Rel$, as follows: $(s_1, s_2) \in R(\Re)$ if, for each $a \in A$,
   i) $s_1 \xrightarrow{<a,n>} s_1'$ implies $s_2 \xrightarrow{<a,n>} s_2'$ such that $(s_1', s_2') \in \Re$;
   ii) $s_2 \xrightarrow{<a,n>} s_2'$ implies $s_1 \xrightarrow{<a,n>} s_1'$ such that $(s_1', s_2') \in \Re$.
2) A relation $\Re \in Rel$ will be called a $R - bisimulation$ if $\Re \subseteq R(\Re)$.
3) We say that two states s_1 and s_2 are *f-performance equivalent*, denoted $s_1 \sim^f s_2$, if and only if there exists a $R - bisimulation$ $\Re$ such that $(s_1, s_2) \in \Re$.
4) We say that two agents E_1, E_2 are *f-performance equivalent*, denoted $E_1 \sim_p^f E_2$, if and only if $0 \Rightarrow E_1 \sim^f 0 \Rightarrow E_2$.

Again, when f is clear from the context, we can omit the superscript f, as in $s_1 \sim s_2$ and $E_1 \sim_p E_2$. This observational semantics equates systems whenever they perform the same actions at the same time thus introducing a simple form of performance evaluation in process algebra.

Proposition 4. Performance equivalence is a congruence on P.

3.2 The Performance Preorder

In order to relate processes with respect to speed, we propose a preorder $\sqsubseteq_p^f$ over P: $E_1 \sqsubseteq_p^f E_2$ if and only if E_1 and E_2 have the same functional behavior but E_1 is at least as fast as E_2 according to duration function f.

Definition 5. (*Performance Preorder*)

1) Let Rel denote the set of binary relations over S. The functional $G : Rel \rightarrow Rel$ is defined, for each $\Re \in Rel$, as follows: $(s_1, s_2) \in G(\Re)$, if for each $a \in A$,
   i) $s_1 \xrightarrow{<a,n>} s_1'$ implies $s_2 \xrightarrow{<a,m>} s_2'$ such that $n \leq m$ and $(s_1', s_2') \in \Re$;
   ii) $s_2 \xrightarrow{<a,n>} s_2'$ implies $s_1 \xrightarrow{<a,m>} s_1'$ such that $n \geq m$ and $(s_1', s_2') \in \Re$.
2) A relation $\Re \in Rel$ will be called a $G - bisimulation$ if $\Re \subseteq G(\Re)$.
3) We say that two states s_1 and s_2 are in the relation of *f-performance preorder* (denoted $s_1 \sqsubseteq^f s_2$) if and only if there exists a $G - bisimulation$ $\Re$ such that $(s_1, s_2) \in \Re$.
4) We say that $E_1 \sqsubseteq_p^f E_2$ if and only if $0 \Rightarrow E_1 \sqsubseteq^f 0 \Rightarrow E_2$.

Whenever clear from the context we will omit the superscript duration function. Hence, we will often write $s_1 \sqsubseteq s_2$ or $E_1 \sqsubseteq_p E_2$.

Example 1. A few examples are now in order.

$$\begin{array}{rlrl} a \mid b \sqsubseteq_p a.b + b.a, & & a.b + b.a \not\sqsubseteq_p a \mid b \\ a \mid b + a.b + b.a \sqsubseteq_p a.b + b.a, & & a.b + b.a \not\sqsubseteq_p a \mid b + a.b + b.a \\ a \mid b \sqsubseteq_p a \mid b + a.b + b.a, & & a \mid b + a.b + b.a \not\sqsubseteq_p a \mid b \end{array}$$

Relation $\sqsubseteq_p$ is reflexive and transitive but not symmetric as the examples above show. When relating processes with respect to speed by means of $\sqsubseteq_p$, of each set of functionally-equivalent computations we focus only on the fastest one and on the slowest one, which provide a lower and upper bound to the time needed for executions. For instance, in $a \mid b + a.b + b.a$ there are two different computations for the sequence ab, with different temporal behaviour. We have that $E_1 = a \mid b + a.b + b.a \sqsubseteq_p E_2 = a.b + b.a$ because the fastest computation for ab in E_1 is faster than the fastest (actually unique) ab computation from E_2, as well as the slowest computation for ab is faster than the slowest ab computation from E_2. All the "performance-intermediate" functionally-equivalent computations are irrelevant, as the following example shows.

Example 2. Consider the following two processes $E_1 = a.a.a + a|a|a$ and $E_2 = a.a.a + a|a|a + a.a|a$. We have that $E_1 \sqsubseteq_p E_2$ and $E_2 \sqsubseteq_p E_1$. This shows that the "intermediate" aaa computations due to subagent $a.a|a$ have no influence on the preorder.

Proposition 6. Performance preorder is a precongruence over P.

Relation $\sqsubseteq_p^f$ depends on the choice of the duration function f, as the following example shows.

Example 3. Consider $E_1 = a.a \mid c$ and $E_2 = a.a \mid c + c.(a \mid a)$. We can show that $E_1 \sqsubseteq_p^f E_2$ if and only if $f(a) \leq f(c)$. In fact, every transition of $0 \Rightarrow E_1$ can be matched (in the same time) by a transition of $0 \Rightarrow E_2$. Similarly, transitions performed by the summand $0 \Rightarrow a.a \mid c$ of E_2 can be matched by transitions of $0 \Rightarrow E_1$. Consider now the transitions from $0 \Rightarrow c.(a \mid a)$. This term can perform $0 \Rightarrow c.(a \mid a) \xrightarrow{<c,f(c)>} f(c) \Rightarrow a \mid f(c) \Rightarrow a$ and, in order to match this transition, $0 \Rightarrow E_1$ can only perform $0 \Rightarrow E_1 \xrightarrow{<c,f(c)>} 0 \Rightarrow a.a \mid f(c) \Rightarrow nil$. Now, $0 \Rightarrow a.a \mid f(c) \Rightarrow nil \sqsubseteq^f f(c) \Rightarrow a \mid f(c) \Rightarrow a$ if and only if $f(a) \leq f(c)$ because the second a of the left state will be completed after $2 \times f(a)$ units of time, while the second a of the right one after $f(c) + f(a)$.

3.3 Competitive Equivalence

In a natural way, the kernel of the preorder gives rise to a notion of equivalence, called *competitive equivalence.*

Definition 7. Two agents E_1 and E_2 are *competitive* equivalent (denoted $E_1 \simeq_p E_2$) if $E_1 \sqsubseteq_p E_2$ and $E_2 \sqsubseteq_p E_1$. In other words, $\simeq_p = \sqsubseteq_p \cap \sqsupseteq_p$.

Analogously to performance preorder, competitive equivalence discriminates agents only by considering the fastest computation and the slowest one for any set of functionally-equivalent computations. Two agents are equivalent if these upper and lower bounds coincide. Also in this case, all the "performance-intermediate" functionally-equivalent computations are irrelevant, as Example 2 shows: $E_1 \simeq_p E_2$ even if E_2 has additional "intermediate" aaa computations.

Relation $\simeq_p$ enjoys the same properties of congruence enjoyed by the preorder. In particular it is preserved by every syntactic construct of the language, including recursion. Moreover, similarly to the preorder, competitive equivalence does depend on the duration assigned to every action.

Proposition 8. Let E_1 and E_2 be two agents and f be a duration function. Then $E_1 \simeq_p^f E_2$ does not imply $E_1 \simeq_p^g E_2$ for any action duration function g.

Proof. Consider processes $E_1 = a.a \,|\, c + c.a.a$ and $E_2 = a.a \,|\, c + c.a.a + c.(a \,|\, a)$. We can show that $E_1 \simeq_p^f E_2$ if and only if $f(a) \leq f(c)$. To prove $E_1 \simeq_p^f E_2$ we have to show both $E_1 \sqsubseteq_p^f E_2$ and $E_2 \sqsubseteq_p^f E_1$. The latter immediately follows whatever is the time assigned to actions a and c. The former follows if and only if $f(a) \leq f(c)$ by similar arguments of Example 3.

The same does not happen for the specific case of constant durational functions, as stated in the following proposition.

Proposition 9. Let E_1 and E_2 be two agents and f, g be any two constant duration functions. Then $E_1 \simeq_p^f E_2$ implies $E_1 \simeq_p^g E_2$.

We now compare performance and competitive equivalences. Because $=$ implies $\leq$ and $=$ implies $\geq$ in N^+, $\sim_p$ is indeed a $G - bisimulation$.

Proposition 10. For E_1, $E_2 \in P$ and any f, we have that $E_1 \sim_p^f E_2$ implies $E_1 \simeq_p^f E_2$.

The converse implication does not hold, i.e. $\simeq_p^f$ is strictly weaker than $\sim_p^f$. To this, consider the two agents in Example 2, which are $\simeq_p^f$-equivalent but not $\sim_p^f$-equivalent.

4 Comparative Performance Semantics

In this section we first show how our preorder relates with *strong bisimulation equivalence*. Then we compare our preorder with *causal preorder* [1] and *location preorder* [4]. Finally, we establish how competitive equivalence relates with the best known untimed non-interleaving semantics proposed in the literature, namely *location* [4], *causal* [6], *ST* [9] and *step* [13] equivalences.

4.1 Relationships with Strong Bisimulation Equivalence

Performance preorder is related to bisimulation equivalence $\sim$ as illustrated in the following theorem (where P_{seq} is the set of processes built without parallel composition).

Theorem 11. Let $E_1 \in P_{seq}$, E_2, $E_3 \in P$ and f is any duration function. Then:

1) $E_1 \sim E_2$ iff $E_2 \sqsubseteq_p^f E_1$.
2) $E_2 \sqsubseteq_p^f E_3$ implies $E_2 \sim E_3$,

4.2 Relationships with Non-interleaving Preorders

We first consider the case of the causal preorder [1]. In this semantic theory, transitions are of the form $E_1 \xrightarrow{u} E_2$ where E_1 and E_2 are P terms, while u is a pomset of actions in Act, called computation. A preorder $\leq_C$ over computations/pomsets is defined: $u \leq_C v$ if u and v correspond to the execution of the same actions, but performed in a less caused fashion in the computation v. The syntax preorder $\leq_C$ is used to induce a preorder, called causal preorder and denoted $\sqsubseteq_c$, over the set of processes P. This preorder preorder does not reduce concurrency to sequentiality and nondeterminism. Its definition is very similar to that of performance preorder but instead of relating natural numbers, it relates pomsets with respect to $\leq_C$. The following proposition shows that $\sqsubseteq_c$ is incomparable with our performance preorder.

Proposition 12. Let E_1 and E_2 be two P terms. Then:

i) $E_1 \sqsubseteq_c E_2$ does not imply $E_1 \sqsubseteq_p^f E_2$ iii) $E_1 \sqsubseteq_p^f E_2$ does not imply $E_1 \sqsubseteq_c E_2$
ii) $E_1 \sqsubseteq_c E_2$ does not imply $E_1 \sqsupseteq_p^f E_2$ iv) $E_1 \sqsubseteq_p^f E_2$ does not imply $E_1 \sqsupseteq_c E_2$

Proof. We provide a pair of processes related by $\sqsubseteq_c$ but not by $\sqsubseteq_p^f$ or $\sqsupseteq_p^f$ and vice versa. i) $a.b + b.a \sqsubseteq_c a \mid b$ but $a.b + b.a \not\sqsubseteq_p^f a \mid b$ for any duration function f. ii) $a.(b+c)+a \mid b \sqsubseteq_c a.(b+c)+a \mid b+a.b$ but $a.(b+c)+a \mid b \not\sqsupseteq_p^f a.(b+c)+a \mid b+a.b$ for any duration function f. iii) $a \mid b \sqsubseteq_p^f a.b + b.a$ for any f but $a \mid b \not\sqsubseteq_c a.b + b.a$. vi) $a.a|a.a \sqsubseteq_p^f a.a.(a|a)$ for any f, but $a.a.(a|a) \not\sqsubseteq_c a.a|a.a$. It is clear that $a.a|a.a \sqsubseteq_p^f a.a.(a|a)$. To show that $a.a.(a|a) \not\sqsubseteq_c a.a|a.a$ consider the transitions $a.a|a.a \xrightarrow{a.a} nil|a.a \xrightarrow{a.a} nil|nil$ which cannot be successfully matched by $a.a.(a|a)$.

For similar reasons the kernel of the causal preorder, $\simeq_c$, does not coincide with our competitive equivalence. Processes $E_1 = a.(b + c) + a|b + a.b$ and $E_2 = a.(b+c)+a|b$ are $\simeq_c$-equivalent but not competitive equivalent (note that $E_1 \not\sqsubseteq_p^f E_2$). To prove that $\simeq_p^f$ does not imply $\simeq_c$, consider the pair of processes in Proposition 8 which are $\simeq_p^f$-equivalent (for $f(a) \leq f(c)$) but not $\simeq_c$-equivalent. It follows that $\simeq_p^f$ and $\simeq_c$ are incomparable.

Let us now consider location preorder from [4]. The intuition behind this preorder is that two processes are related if the first is "more distributed" over the space than the other one. Technically, location names (taken from an infinite set Loc of names) are dynamically assigned to performed actions. This is achieved by means of a transition relation of the form $\xrightarrow{a,u}$, where a is an action, $u = l_1...l_n l$ is a word over Loc. For u, $v \in Loc^*$, $u \leq v$ holds if u is a subword of v. This preorder on location names induces a preorder $\sqsubseteq_l$ on processes. Its definition is very similar to that of performance preorder but instead of relating natural numbers, it relates locations with respect to $\leq$. We state that location preorder strictly implies performance preorder so that if E_1 is "more distributed than" E_2, then E_1 is "faster than" E_2, but the vice versa is not always true.

Proposition 13. Let E_1 and E_2 be in P.Then, $E_1 \sqsubseteq_l E_2$ implies $E_1 \sqsubseteq_p E_2$.

For the counter-example, let us consider the pair of processes in Example 3. They are $\sqsubseteq_p^f$-related if $f(a) \leq f(c)$, but they are not $\sqsubseteq_l$-related.

The same holds when considering the kernel of location preorder (defined as usual by $\simeq_l = \sqsubseteq_l \cap \sqsupseteq_l$) and competitive equivalence.

Proposition 14. Let E_1 and E_2 be in P. Then, $E_1 \simeq_l E_2$ implies $E_1 \simeq_p E_2$.

4.3 Relationships with Non-interleaving Equivalences

Proposition 15. For processes in P, location equivalence as well as causal equivalence and ST equivalence strictly imply competitive equivalence.

It is worth noting that competitive equivalence is incomparable with *step equivalence*, the coarsest non-interleaving semantics proposed so far. Intuitively the transitions of the step semantics are labelled by multisets of actions which are concurrently executable. Thus for example $a|b \sim_{Step} a|b + a.b$. Clearly $a|b \not\simeq_p^f a|b + a.b$ for any f because $a|b + a.b \not\sqsubseteq_p^f a|b$. To see that $\simeq_p^f$ does not imply $\sim_{Step}$ consider the pair of processes E_1 and E_2 in Proposition 8 which are $\simeq_p^f$-equivalent (for $f(a) \leq f(c)$) but not $E_1 \sim_{Step} E_2$. In fact process $E_2 \xrightarrow{c} a|a$ while E_1 can perform c but the resulting two a actions can be performed sequentially only.

5 Conclusions

In this paper we presented a semantic theory for a process description language which is based on a preorder $\sqsubseteq_p$ relating two processes E_1 and E_2 if they have the same functional behaviour and E_1 is at least as fast as E_2. We have shown that performance preorder and its kernel equivalence can be soundly and completely axiomatized over P_{fin}. Finally, we have proved the intutive and expected fact that the more distributed a process is, the more efficient it is. In [5] a complete and sound axiomatization of performance preorder and competitive equivalence is provided, together with an example of stepwise refinement of a specification to its optimal implementation.

5.1 Related Work

Two preorders relating processes with respect to speed have been proposed in the literature. Both have in common with us only the goal because they conceive "time passing" in a completely different way. In [3] a notion of efficiency is based on the number of communications that have taken place. Hence, E_1 is "faster" than E_2 if E_1 never need to perform more internal (τ-) actions than E_2. This notion has an intuitive justification in the fact that communications are often the most expensive operations to be performed. However, no performance measure is available for a language without communication. In [12] a "faster than" relation has been studied in the so-called abstract time approach: actions are atomic and time passes "in between" them. We claim that this approach to timed agents, even if appropriate for specific applications, is less natural than ours as only in our approach one can formally prove the basically intutive statement that "the more distributed a process is, the more efficient it is".

5.2 Open Problems and Future Research

Extending our theory to communicating processes seems not trivial. In the setting of durational actions, two different synchronization rules have been proposed [2, 8]. However, in both cases performance preorder is not a precongruence for parallel composition with synchronization. Further work will be devoted to solve this problem. Another, quite interesting line of research is the study of "optimal implementations": given a specification E_1, find the fastest E_2 which is functionally equivalent to E_1. This means that we would like to study algorithms for finding the minimal elements in the performance preorder. Due to the comparison with the location preorder, it seems clear that this problem is tightly related to the problem of finding *parallel decompositions* of processes, where the notion of decomposition we have in mind has some connection with the one in [11].

Acknowledgements

The first author thanks Rocco De Nicola for his encouragement to explore this area of research.

References

1. L.Aceto, *On Relating Concurrency and Nondeterminism*, Technical Report SI/RR–89/06, University of Sussex, 1989.
2. L.Aceto, D.Murphy, *Timing and Causality in Process Algebra*, Technical Report 9/93, University of Sussex, 1993, to appear in Acta Informatica.
3. S.Arun-Kumar and M.Hennessy, *An Efficiency Preorder for Processes*, Acta Informatica **29**, pp.737-760, 1992.
4. G.Boudol, I.Castellani, M.Hennessy, A.Kiehn, *A Theory of Processes with Localities*, Formal Aspects of Computing **6**, pp. 165-200, 1993.
5. F.Corradini,, R.Gorrieri and M.Roccetti, *Performance Preorder and Competitive equivalence*, Tech.Rep. LFCS-95-1, University of Bologna, January 1995.
6. Ph.Darondeau, P.Degano, *Causal trees*, In Automata, Languages and Programming, Springer, LNCS **372**, 1989.
7. G-L.Ferrari and U.Montanari, *Observing Time-Complexity of Concurrent Programs*, Technical Report, Dipartimento di Informatica, Università di Pisa, 1993.
8. R.Gorrieri, M.Roccetti and E. Stancampiano, *A Theory of Processes with Durational Actions*, Theoretical Computer Science **140** (1), pp. 73-94, March 1995.
9. R.J. van Glabbeek, F.Vaandrager, *Petri Net Models for Algebraic Theories of Concurrency*, In Proc. of PARLE II, Springer, LNCS **259**, pp. 224-242, 1987.
10. R.Milner, *Communication and concurrency*, International Series on Computer Science, Prentice Hall International, 1989.
11. R.Milner, F.Moller, *Unique Decomposition of Processes.* Theoretical Computer Science **107**, pp. 357-363, 1993.
12. F.Moller, C.Tofts, *Relating Processes with Respect to Speed.* In CONCUR'91, LNCS **527**, Springer-Verlag, pp. 424-438, 1991.
13. M.Nielsen, P.S.Thiagarajan, *Degrees of Nondeterminism and Concurrency*, In Proc. 4th Conf. on FST & TCS, Springer, LNCS **181**, pp. 89-117, 1984.