What is branching time semantics and why to use it?

R.J. van Glabbeek*
Computer Science Department, Stanford University
Stanford, CA 94305, USA.
rvg@cs.stanford.edu

The concept of branching time in the semantics of concurrent systems is well known and well understood. Still a formal definition of what it means for a model or equivalence to respect branching time has never explicitly be given. This note proposes such a definition. Additionally the opportunity is taken to voice an old but poorly understood argument for using branching time semantics instead of models or equivalences that are fully abstract with respect to some notion of observability.

Introduction

When comparing models or equivalences for concurrent systems, it is common practice to distinguish between *linear time* and *branching time* semantics (see for instance De Bakker, Bergstra, Klop & Meyer [1] or Pnueli [9]). In the former, a process is completely determined by the observable content of its possible (partial) runs, whereas in the latter also the information is preserved where two different courses of action diverge (although branching of identical courses of action may still be neglected). Standard examples are the processes in Figure 1 and 2. In Figure 1, both processes

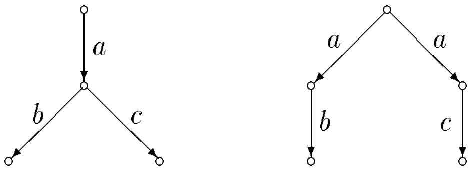

Figure 1: $a(b+c)$ vs. $ab+ac$

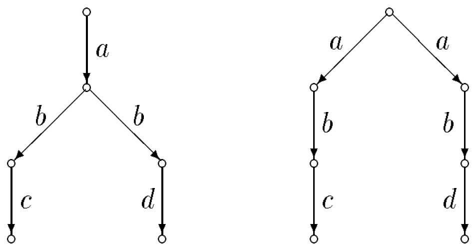

Figure 2: $a(bc+bd)$ vs. $abc+abd$

have two complete runs, whose observable content is ab and ac respectively. However, in $a(b+c)$ these runs diverge after the a-step, whereas in $ab+ac$ they diverge at the onset. This difference can also be described by pointing out that $a(b+c)$ has a partial run with observable content a that is an initial part of each of the runs ab and ac, whereas $a(b+c)$ has no such run. Hence the two processes are linear time equivalent, but not branching time equivalent. Similarly, the two processes of Figure 2 are identified in linear time semantics but not in branching time semantics, since only $a(bc+bd)$ has a run a that is an initial part of both of the runs abc and abd.

Figure 3 illustrates that in branching time semantics only the branching of *different* courses of action is of importance. The processes $a(b+b)$ and $ab+ab$ are *bisimulation equivalent* [8], which is

*This work was supported by ONR under grant number N00014-92-J-1974.

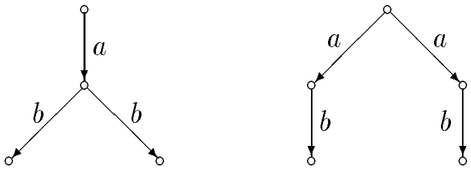

Figure 3: $a(b+b)$ vs. $ab+ab$

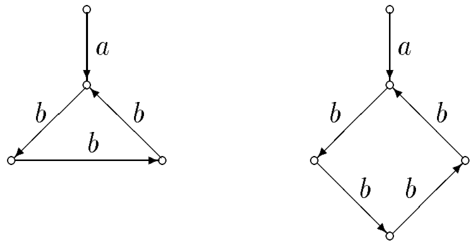

Figure 4: $a(bbb)^{\infty}$ vs. $a(bbbb)^{\infty}$

widely considered to be the prime example of a branching time equivalence. Nevertheless, the two runs with observable content ab diverge at a different state. One could employ a stricter notion of branching time, in which also the branching of identical courses of action would be taken into account. Under such a notion, the processes of Figure 3 should be distinguished, but the ones of Figure 4 could safely be identified.

It should be pointed out that branching time is not the definition of a particular model or semantic equivalence, but a criterion that can be satisfied by models or equivalences. A model or equivalence that satisfies this criterion is said to *be a branching time model or equivalence*, to *respect branching time*, or to *respect the branching structure* of processes. Call two processes, represented as labelled transition systems, *tree equivalent* if their unfoldings into trees are isomorphic. The two processes of Figure 4 for instance are tree equivalent, whereas the ones of Figure 1–3 are not. Tree equivalence is strictly finer than bisimulation equivalence and thus certainly preserves the information on where different courses of action diverge. Hence it constitutes a second example of a branching time equivalence. Moreover, unlike bisimulation equivalence, it would still be a branching time equivalence under the stricter interpretation of branching time contemplated above.

For processes without silent moves, it is commonly agreed that an equivalence respects branching time iff it is finer than or equal to bisimulation equivalence. Hence this yield a candidate for a formal definition of the concept of a branching time equivalence. However, in the presence of silent moves, or τ-steps, the situation is less clear. In VAN GLABBEEK & WEIJLAND [7] it is argued that *branching bisimulation*, that in the absence of silent moves coincides with plain bisimulation, takes over this rôle of bisimulation in the presence of τ's. However, this is not inherent in the definition of the equivalence, and was meant to be a result rather than a definition. The reasoning of [7] is illustrated in Figures 5 and 6. The first process in Figure 5 has a run with observable content ab,

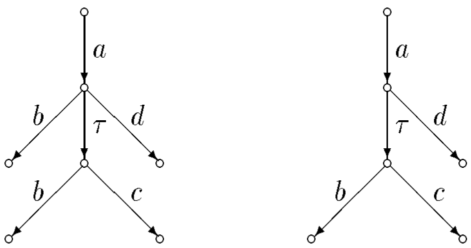

Figure 5: $a(b+\tau(b+c)+d)$ vs. $a(\tau(b+c)+d)$

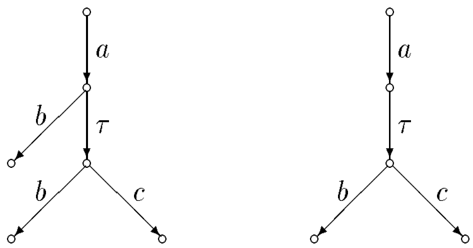

Figure 6: $a(b+\tau(b+c))$ vs. $a\tau(b+c)$

that diverges from the run with observable content ac at the same point where it diverges from the run ad. Thus it does not pass through a state in which it is still possible to continue with a course of action that involves a c, but not possible to continue with a course of action involving a

d. Such a run is not present in the other process. It follows that these processes have a different branching structure and can not be identified by a branching time equivalence. As MILNER's notion of *weak bisimulation* [8] identifies both processes, it does not respect branching time. The notion of branching bisimulation on the other hand gives rise to a finer equivalence that does distinguish the processes of Figure 5. The processes of Figure 6 are branching bisimulation equivalent however, and indeed no argument can be construed that would indicate disrespect of branching time, the key argument being that the various runs with observable content ab can be regarded as the same course of action.

The best definition of a branching time equivalence so far is perhaps the one implicit in the notion of a *consistent colouring*, introduced in [7]: An equivalence *respects branching time* if the colouring on processes/states, obtained by giving equivalent processes the same colour, is *consistent* in the sense of [7]. Here a sequence $C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k$ is a *concrete coloured trace* of a process p_0 if there is a run $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \longrightarrow \cdots \xrightarrow{a_k} p_k$, such that process p_i has colour C_i; a *coloured trace* of p is a sequence $C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k$ obtained from a concrete coloured trace of p by replacing any subsequence $C, \tau, C, \tau, ..., \tau, C$ by C; and a colouring is consistent if processes with the same colour have the same set of coloured traces. The idea behind a consistent colouring is that processes with the same colour have the same potential of further courses of action. A coloured trace indicates how these potentials vary (diminish) during a run. As expected, under this definition branching bisimulation turns out to be the coarsest branching time equivalence.

Although this definition captures the concept of branching time reasonably well, it has the disadvantage of being tailored to a setting with silent moves (through the reduction $C, \tau, C, \tau, ..., \tau, C \rightsquigarrow C$). What would be more convincing would be a definition that in no way refers to internal actions, and still yields branching bisimulation as the coarsest equivalence respecting the branching structure of processes. Moreover, it would be nice to know what precisely the branching structure of a process is. This is the goal of the present paper. I will formally define the branching structure of a process, and declare an equivalence to respect branching time iff it only relates processes with the same branching structure. This will be the case iff the equivalence is finer than or equal to branching bisimulation equivalence. Similarly a model of concurrency respects branching time iff processes that are represented by the same semantic object have the same branching structure.

1 The branching structure of a process

Let $\mathbb{P}$ be a class of processes. I assume that for each process $p \in \mathbb{P}$ a set $run(p)$ of (partial) runs is defined, equipped with a prefix ordering $\leq$, and that with each run (or *execution*) $e \in run(p)$ is associated its observable content $l(e)$.

Definition 1 (*Pomset*). A *labelled partial ordered set (elpo)* is a triple $(E, \leq, l)$ with E a set, $\leq$ a partial order on E, and l a function with domain E. Two elpos $(E, \leq_E, l_E)$ and $(F, \leq_F, l_F)$ are *isomorphic* if there exists a bijective function $i : E \rightarrow F$ (an *isomorphism*) such that, for $e, e' \in E$, $e \leq_E e' \Leftrightarrow i(e) \leq_F i(e')$ and $l_F(i(e)) = l_E(e)$. Of course isomorphism ($\cong$) is an equivalence relation, and an equivalence class of isomorphic elpos is called a *partial ordered multiset* or *pomset*.

A first idea of what could be the branching structure of a process $p \in \mathbb{P}$, is the pomset with representative $(run(p), \leq, l)$. This I will call the *strict branching structure* of p, as it corresponds with the idea that the branching of all, possibly indistinguishable, runs is taken into account. In order to arrive at a less strict definition, one needs to identify certain runs.

Definition 2 (*Congruence*). If $\sim$ is an equivalence relation on a set E, then $E/_\sim$ denotes the set of equivalence classes of $\sim$ and $[e]_\sim$ denotes the equivalence class containing $e \in E$.

A *congruence relation* on an elpo $(E, \leq, l)$ is an equivalence relation $\sim$ on E, such that

$$e \sim f \Rightarrow l(e) = l(f) \quad \text{and} \quad \exists e'(e \leq e' \sim f') \Leftrightarrow \exists f(e \sim f \leq f').$$

If $\sim$ is a congruence on an elpo $(E, \leq, l)$, then $\leq_\sim$ and $l_\sim$ are defined on $E/_\sim$ by

$$[e]_\sim \leq_\sim [f']_\sim \Leftrightarrow \exists e' \in [f']_\sim(e \leq e') \left(\Leftrightarrow \exists f \in [e]_\sim(f \leq f') \right) \quad \text{and} \quad l_\sim([e]_\sim) = l(e).$$

Note that these definitions are independent of the choice of representatives $e \in [e]_\sim$ and $f' \in [f']_\sim$.

This is the intermediate variant of three possible generalizations of the concept of a congruence from algebras to algebraic structures with relations. A *strong* congruence would even require that $e \sim f \wedge e' \sim f' \Rightarrow (e \leq e' \Leftrightarrow f \leq f')$, so that $\leq_\sim$ can simply be defined by $[e]_\sim \leq_\sim [e']_\sim \Leftrightarrow e \leq e'$. A *weak* congruence on the other hand doesn't require anything for $\leq$, and $\leq_\sim$ needs to be defined by $[e]_\sim \leq_\sim [e']_\sim \Leftrightarrow \exists f \in [e]_\sim, f' \in [e']_\sim(f \leq f')$.

Definition 3 The *branching structure* of a process p is the isomorphism class of $(run(p)/_\sim, \leq_\sim, l_\sim)$, where $\sim$ is the coarsest congruence on $(run(p), \leq, l)$. An equivalence $\equiv$ on $\mathbb{P}$ *respects the branching structure of processes* if $p \equiv q \Rightarrow [(run(p)/_\sim, \leq_\sim, l_\sim)]_\cong = [(run(q)/_\sim, \leq_\sim, l_\sim)]_\cong$.

Obviously a coarsest congruence on a pomset exists, as the union of arbitrary many congruences is again a congruence. The following observations are meant as tool for deciding whether or not a model or equivalence respects branching time.

Lemma 1 Two processes $p, q \in \mathbb{P}$ have the same branching structure iff there exists congruences $\sim_p$ and $\sim_q$ such that $(run(p)/_{\sim_p}, \leq_{\sim_p}, l_{\sim_p}) \cong (run(q)/_{\sim_q}, \leq_{\sim_q}, l_{\sim_q})$.

Proof: This follows immediately from the following two facts, whose proofs are straightforward.

- If $\sim$ is the coarsest and $\sim_1$ any congruence on an elpo $(E, \leq, l)$, and $\sim_2$ is the coarsest congruence on $(E/_{\sim_1}, \leq_{\sim_1}, l_{\sim_1})$, then $(E/_{\sim_1}/_{\sim_2}, \leq_{\sim_{1\sim_2}}, l_{\sim_{1\sim_2}}) \cong (E/_\sim, \leq_\sim, l_\sim)$.
- If $(E, \leq, l) \cong (F, \leq, l)$ and $\sim$ is a congruence on $(E, \leq, l)$, then there is a congruence $\sim'$ on $(F, \leq, l)$ such that $(E/_\sim, \leq_\sim, l_\sim) \cong (F/_{\sim'}, \leq_{\sim'}, l_{\sim'})$. Moreover, $\sim'$ is the coarsest iff $\sim$ is. □

Proposition 1 Two processes $p, q \in \mathbb{P}$ have the same branching structure iff there exists a relation $R \subseteq run(p) \times run(q)$ with domain $run(p)$ and range $run(q)$, satisfying

$$eRf \Rightarrow l(e) = l(f), \quad \exists e'(e \leq e'Rf') \Leftrightarrow \exists f(eRf \leq f') \quad \text{and} \quad \exists f'(f \leq f'R^{-1}e') \Leftrightarrow \exists e(fR^{-1}e \leq e').$$

Proof: "If": Let R be such a relation. Define $\sim_p$ (and similarly $\sim_q$) by

$$e \sim_p e' \Leftrightarrow e = e_0 R f_1 R^{-1} e_1 R f_2 R^{-1} e_2 R \ldots R f_n R^{-1} e_n = e' \; (n \geq 0).$$

It is easy to see that $\sim_p$ is a congruence relation. Moreover, the function $i : run(p) \to run(q)$ defined by $i([e]_{\sim_p}) = [f]_{\sim_q}$ for some $f \in run(q)$ with eRf, is well defined and an isomorphism.
"Only if": Let $i : run(p)/_\sim \to run(q)/_\sim$ be an isomorphism. Define R by $eRf \Leftrightarrow i([e]_\sim) = [f]_\sim$. Again it is straightforward to check that R has the required properties. □

2 The case of labelled transition spaces with and without τ

In this section I will instantiate the general definition of branching time from the previous section for equivalences on models of concurrency that can be regarded as labelled transition spaces.

Definition 4 A *labelled transition space (LTS)* is a pair $(\mathbb{P}, \rightarrow)$ with $\mathbb{P}$ a class (of *processes*) and $\rightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$ for Act a set (of *actions*), such that for $p \in \mathbb{P}$ and $a \in Act$ the class $\{q \in \mathbb{P} \mid p \xrightarrow{a} q\}$ is a set.

Notation: Write $p \xrightarrow{a} q$ for $(p, a, q) \in \rightarrow$ and $p \xrightarrow{a}$ for $\exists q \in \mathbb{P} : p \xrightarrow{a} q$.

The elements of $\mathbb{P}$ represent the processes we are interested in, and $p \xrightarrow{a} q$ means that process p can evolve into process q while performing the action a. I use the word 'space' instead of 'system' to emphasize that the elements of an LTS are *all* systems under investigation, and not merely the states of a single system.

Definition 5 (*Run*). A sequence of transitions $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \cdots p_{n-1} \xrightarrow{a_n} p_n$ for $n \in \mathbb{N}$ in an LTS $(\mathbb{P}, \rightarrow)$ is called a *run* of p_0. Let $run(p)$ be the set of runs of a process $p \in \mathbb{P}$, and let $\leq$ be the prefix ordering on runs.

It remains to determine the observable content of a run. In case all actions are observable, let $l(p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \cdots p_{n-1} \xrightarrow{a_n} p_n) = a_1 a_2 \cdots a_n$. In case $Act = A \dot{\cup} \{\tau\}$, where τ denotes an internal or unobservable action, the observable content $l(e)$ of a run e is the same sequence, but from which all τ's are removed. This completes the definition of the branching structure of members of an LTS.

Definition 6 (*Back and forth bisimulation*, De Nicola, Montanari & Vaandrager [5]). Let $(\mathbb{P}, \rightarrow)$ be an LTS. Two processes $p, q \in \mathbb{P}$ are *weakly back and forth bisimilar* if there exists a relation $R \subseteq run(p) \times run(q)$, called a *weak back and forth bisimulation*, satisfying

- $\lambda R \lambda$;
- if eRf and $e \stackrel{\sigma}{\Longrightarrow} e'$ (for $\sigma \in A^*$), then there exists an f' such that $f \stackrel{\sigma}{\Longrightarrow} f'$ and $e'Rf'$;
- if $e'Rf'$ and $e \stackrel{\sigma}{\Longrightarrow} e'$ (for $\sigma \in A^*$), then there exists an f such that $f \stackrel{\sigma}{\Longrightarrow} f'$ and eRf;
- if eRf and $f \stackrel{\sigma}{\Longrightarrow} f'$ (for $\sigma \in A^*$), then there exists an e' such that $e \stackrel{\sigma}{\Longrightarrow} e'$ and $e'Rf'$;
- if $e'Rf'$ and $f \stackrel{\sigma}{\Longrightarrow} f'$ (for $\sigma \in A^*$), then there exists an e such that $e \stackrel{\sigma}{\Longrightarrow} e'$ and eRf.

Here λ denotes the empty run and $e \stackrel{\sigma}{\Longrightarrow} e'$ means $e \leq e' \wedge l(e') = l(e)\sigma$.

Theorem 1 Let $\mathbb{P}$ be an LTS. Two processes $p, q \in \mathbb{P}$ have the same branching structure iff they are weakly back and forth bisimilar.

Proof: Immediately from Proposition 1. □

In [5] is has been established that weak back and forth bisimulation equivalence coincides with branching bisimulation equivalence. Hence is follows that

Corollary 1 A semantic equivalence on a labelled transition space respects the branching structure of processes iff it is finer than or equal to branching bisimulation equivalence. □

If in Definition 3 the coarsest congruence would be replaced by the coarsest *weak* congruence, the resulting 'branching structure' would have a distinct linear time flavour, as it would be nothing more than the so-called *trace set* of a process. The coarsest *strong* congruence on the other hand would on an LTS be the identity relation, causing the branching structure to become the *strict* branching structure. On an LTS two processes have the same strict branching structure iff they are tree equivalent, as follows immediately from the definitions. Silent actions play no special rôle here.

3 Other models

Term models, in which a process is represented as an expression in a system description language, labelled event structures, Petri nets, process graphs, and many other models of concurrency can be regarded as labelled transition spaces. Namely, there are canonical definitions of the transition relations $\xrightarrow{a}$ between two terms, event structures, nets, etc. For these models the definitions of runs and their visible content from the previous section apply, and hence it is determined what it means for a semantic equivalence on such a model to respect branching time. Unless causality is taken to be part of the observable content of runs, in which case $l(e)$ could be modelled as a *mixed ordering* (cf. DEGANO, DE NICOLA & MONTANARI [4]) and the coarsest branching time equivalence would be *history preserving branching bisimulation equivalence*.

Some models however, such as the *failures model* of BROOKES, HOARE & ROSCOE [3], can not be regarded as labelled transition spaces. In these cases it is often difficult to associate with a process a prefix-ordered set of runs. However, usually such a model can be canonically interpreted as the homomorphic image of a labelled transition space. In that case the model can be said to *respect branching time* iff the canonical homomorphism does not map two processes which have a different branching structure to the same element. Similarly, an equivalence on such a model respects branching time iff no two images of processes with a different branching structure are equivalent.

4 What is nice about branching time?

Semantic equivalences (and preorders) for concurrent systems are used as criteria for determining whether implementations (say of a protocol) meet specifications. The choice of a suitable equivalence (or preorder) can be motivated by means of a *testing scenario*. Such a scenario associates to every system a set of 'observable' properties. These properties should be the ones that are important in the given application. Now the equivalence should be such that two processes are related **only** if they have the same observable behaviour (and for preorders the observable properties of the one should be included in those of the other). But this still leaves us with an abundance of suitable equivalences to choose from. Here one is faced with two opposite strategies.

One strategy is to choose the unique equivalence that relates two systems **if** and only if they have the same observable behaviour. This equivalence is said to be *fully abstract* with respect to the selected testing scenario. The advantage of a fully abstract equivalence is that it never fails to identify two processes if they have the same observable behaviour. For this reason it may be the best choice if the testing scenario is known and fixed once and forever. In practice however, there appears to be doubt and difference of opinion concerning the observable behaviour of systems. Moreover, what is observable may depend on the nature of the systems on which the concept will be applied and the context in which they will be operating. A big disadvantage of equivalences that are fully abstract with respect to non-stable notions of observability is that whenever a verification is carried out using a such an equivalence, and one decides that the context in which the verified system will be working is such that actually a little bit more can be observed that what was originally accounted for, the verification has to be completely redone. Moreover, the correctness of the investigated systems keeps depending on the completeness of the underlying testing scenario.

The opposite strategy is based on a concept of 'internal structure' of processes. The internal structure of a process should be such that for any reasonable notion of observability you can imagine, if two processes have the same internal structure they surely have the same observable behaviour. According to this strategy a suitable equivalence should respect the internal structure of processes: if two processes are equivalent they must have the same internal structure. Preferably the equivalence should be 'fully abstract' with respect to this structure: processes with a different internal structure should be distinguished. This to maximize the applicability of the notion. A disadvantage of this strategy is that the selected equivalence may fail to identify two processes with the same observable behaviour merely because they have a different internal structure. But the advantage is that verifications (of the equivalence of two processes) based on this strategy automatically count as verifications in any equivalence that is fully abstract with respect to a testing scenario, and keep being valid if the insight in what is observable changes. Moreover, when applying such an equivalence all bothersome considerations about observability can be skipped.

In models of concurrency that abstract from divergence, true concurrency, real-time behaviour and stochastic aspects of systems, and represent actions by uninterpreted symbols, it appears that the internal structure of processes can be formalized as their branching structure. This makes branching bisimulation equivalence a preferred tool for verifications. The internal structure strategy recommends to try a verification first in branching bisimulation semantics, and only when this fails move to a coarser equivalence that still seems appropriate for the given application. In this move to a coarser equivalence it would still be recommendable to minimize the amount of water (linear time) in the wine. But of course in situations where the appropriate testing scenario is beyond doubt, the full abstraction strategy is recommendable, and would rarely yield so fine a semantics as branching bisimulation.

One could argue that the strict branching structure would be an even better formalization of the internal structure of processes. This would make tree equivalence a preferred option. Although this argument by itself has some merit, the use of tree semantics has serious drawbacks. To name a few, the standard operational and denotational semantics of CCS-like system description languages do not coincide module tree equivalence, and deciding tree equivalence of regular processes has a higher complexity than deciding branching bisimulation equivalence. But most importantly, whereas in many verifications the specification and implementation are actually branching bisimulation equivalent, it rarely occurs that they are tree equivalent.

Another counterargument could be that there is an equivalence coarser than branching bisimulation that, although not respecting the branching structure of processes, respects their internal structure to a sufficient degree to guarantee that any reasonable testing scenario yields an equivalence that is at least as coarse. The most likely candidates for such an equivalence are weak bisimulation or, as argued in the absence of silent moves in BLOOM, ISTRAEL & MEYER [2], *ready simulation*. However, in VAN GLABBEEK [6] I present a testing scenario for branching bisimulation that is arguably only twice as contrived as that of weak bisimulation. Moreover, I argue that in the presence of silent moves the case for ready simulation is not so compelling as in the τ-free situation, and present a situation in which a more discriminating equivalence is called for.

In models of concurrency that do not abstract from true concurrency etc. the internal structure of a process may include more than just its branching structure. The argument presented here can just as well be used to prefer causality respecting semantics for instance.

References

[1] J.W. de Bakker, J.A. Bergstra, J.W. Klop & J.-J.Ch. Meyer (1984): *Linear time and branching time semantics for recursion with merge.* *Theoretical Computer Science* 34, pp. 135–156.

[2] B. Bloom, S. Istrail & A.R. Meyer (1988): *Bisimulation can't be traced: Preliminary report.* In *Conference Record of the* 15^{th} *ACM Symposium on Principles of Programming Languages*, San Diego, California, pages 229–239. Full version available as Technical Report 90-1150, Department of Computer Science, Cornell University, Ithaca, New York, August 1990. Accepted to appear in *Journal of the ACM.*

[3] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes.* *Journal of the ACM* 31(3), pp. 560–599.

[4] P. Degano, R. De Nicola & U. Montanari (1989): *Partial orderings descriptions and observations of nondeterministic concurrent processes.* In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: *REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Noordwijkerhout, LNCS 354, Springer-Verlag, pp. 438–466.

[5] R. De Nicola, U. Montanari & F.W. Vaandrager (1990): *Back and forth bisimulations.* In J.C.M. Baeten & J.W. Klop, editors: *Proceedings CONCUR 90,* Amsterdam, LNCS 458, Springer-Verlag, pp. 152–165.

[6] R.J. van Glabbeek (1993): *The linear time - branching time spectrum II; the semantics of sequential systems with silent moves.* Preliminary version available from `boole.stanford.edu`. Extended abstract to appear in *Proceedings CONCUR 93*, Hildesheim, Germany, LNCS, Springer-Verlag.

[7] R.J. van Glabbeek & W.P. Weijland (1989): *Branching time and abstraction in bisimulation semantics (extended abstract).* In G.X. Ritter, editor: *Information Processing 89*, North-Holland, pp. 613–618. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

[8] R. Milner (1990): *Operational and algebraic semantics of concurrent processes.* In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag, 1980.

[9] A. Pnueli (1985): *Linear and branching structures in the semantics and logics of reactive systems.* In W. Brauer, editor: *Proceedings* 12^{th} *ICALP,* Nafplion, LNCS 194, Springer-Verlag, pp. 15–32.