Branching Time and Abstraction in Bisimulation Semantics

ROB J. VAN GLABBEEK AND W. PETER WEIJLAND

Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands

Abstract. In comparative concurrency semantics, one usually distinguishes between *linear time* and *branching time* semantic equivalences. Milner's notion of *observation equivalence* is often mentioned as the standard example of a branching time equivalence. In this paper we investigate whether observation equivalence really does respect the branching structure of processes, and find that in the presence of the unobservable action τ of CCS this is not the case.

Therefore, the notion of *branching bisimulation equivalence* is introduced which strongly preserves the branching structure of processes, in the sense that it preserves computations together with the potentials in all intermediate states that are passed through, even if silent moves are involved. On closed CCS-terms branching bisimulation congruence can be completely axiomatized by the single axiom scheme:

$$a.(\tau.(y + z) + y) = a.(y + z)$$

(where a ranges over all actions) and the usual laws for strong congruence.

We also establish that for sequential processes observation equivalence is not preserved under refinement of actions, whereas branching bisimulation is.

For a large class of processes, it turns out that branching bisimulation and observation equivalence are the same. As far as we know, all protocols that have been verified in the setting of observation equivalence happen to fit in this class, and hence are also valid in the stronger setting of branching bisimulation equivalence.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation; F.3.2 [**Logics and Meaning of Programs**]: Semantics of Programming Languages; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

---

This paper appeared before as Chapter III of van Glabbeek [1990a]. The first two sections partly appeared in Weijland [1989], an extended abstract appeared as van Glabbeek and Weijland [1989a] and Section 6 as van Glabbeek and Weijland [1989b].

The research of the authors was supported by ESPRIT project 432 (METEOR). R. J. van Glabbeek was also supported by Sonderforschungsbereich 342 of the TU München, by the ESPRIT Basic Research Action 3148 (DEMON) and by Office of Naval Research (ONR) under grant number N00014-92-J-1974; W. P. Weijland was supported by ESPRIT project 3006 (CONCUR). This paper was finalized when R. J. van Glabbeek was employed at the Technical University of Munich and revised when he was employed at Stanford University.

Authors' addresses: R. J. van Glabbeek, Computer Science Department, Stanford University, Stanford, CA 94305-9045, e-mail: rvg@cs.stanford.edu; W. P. Weijland, PTT Telecom, Postbus 30150, 2500 GD's Gravenhage, The Netherlands, e-mail: w.p.weijland@bd.telecom.ptt.nl.

Additional Key Words and Phrases: Abstraction, action refinement, bisimulation, branching time, concurrency, process algebra, semantic equivalence

1. *Introduction*

When comparing semantic equivalences for concurrency, it is common practice to distinguish between *linear time* and *branching time* equivalences (see, for instance, De Bakker, et al. [1984], and Pnueli [1985]). In the former, a process is determined by its possible executions, whereas in the latter also the branching structure of processes is taken into account. The standard example of a linear time equivalence is *trace equivalence* as employed in Hoare [1980]; the standard example of a branching time equivalence is *observation equivalence* or *bisimulation equivalence* as defined by Milner [1980] and Park [1981] (cf. Milner [1983], Milner [1989]). Between pure linear time and branching time equivalences there are several *decorated trace equivalences* (cf. Baeten et al. [1987b], Bloom et al. [1995], Brookes et al. [1984], De Nicola and Hennessy [1984], Hoare [1985], Olderog and Hoare [1986], Phillips [1987], Pnueli [1985], Pomello [1986]), preserving part of the branching structure of processes but for the rest resembling trace equivalence.

Originally, the most popular argument for employing branching time semantics was the fact that it allows a proper modelling of *deadlock behavior*, whereas linear time semantics does not. However, this advantage is shared with the decorated trace semantics, which have the additional advantage of only distinguishing between processes that can be told apart by some notion of *observation* or *testing*. The main criticism on observation equivalence—and branching time equivalences in general—is that it is not an observational equivalence in that sense: distinctions between processes are made that cannot be observed or tested, unless observers are equipped with extraordinary abilities like that of a copying facility together with the capability of global testing as in Milner [1980; 1981] or Abramsky [1987].

Nevertheless, branching time semantics is of fundamental importance in concurrency, exactly because it is independent of the precise nature of observability. Which one of the decorated trace equivalences provides a suitable modelling of observable behavior depends in great extent on the tools an observer has to test processes. And, in general, a protocol verification in a particular decorated trace semantics, does not carry over to a setting in which observers are a bit more powerful. On the other hand, branching time semantics preserves the internal branching structure of processes and thus certainly their observable behavior as far as it can be captured by decorated traces. A protocol verified in branching time semantics is automatically valid in each of the decorated trace semantics.

Probably one of the most important features in process algebra is that of abstraction, since it provides us with a mechanism to *hide* actions that are not observable, or not interesting for any other reason. By abstraction, some of the actions in a process are made *invisible* or *silent*. Consequently, any consecutive execution of hidden steps cannot be recognized since we simply do not 'see' anything happen.

Algebraically, in ACP$_\tau$ of Bergstra and Klop [1985] abstraction has the form of a renaming operator which renames actions into a silent move called τ. In

Milner's CCS (Milner [1980]), these silent moves result from synchronization. This new constant τ is introduced in the algebraic models as well: for instance in the *graph models* (cf. Bergstra and Klop [1985] and Milner [1980]) we find the existence of τ-edges, and so the question was how to find a satisfactory extension of the original definition of *bisimulation equivalence* that we had on process graphs without τ.

It turns out that there exist many possibilities for extending bisimulation equivalence to process graphs with τ-steps. One such possible extension is incorporated in Milner's notion of *observation equivalence*—also called *weak bisimulation equivalence*—which resembles ordinary bisimulation, but permits arbitrary sequences of τ-steps to precede or follow corresponding atomic actions. A different notion, the so-called *η-bisimulation*, was suggested by Baeten and Van Glabbeek [1987], yielding a weaker set of abstraction axioms. In Milner [1981], another notion of observational equivalence was introduced which in this paper is referred to as *delay bisimulation equivalence*. As we will show, the treatments of Milner and Baeten & Van Glabbeek fit into a natural structure of four possible variations of bisimulation equivalence involving silent steps. The structure is completed by defining *branching bisimulation equivalence*. As it turns out, observation equivalence is the coarsest equivalence of the four, in the sense of identifying more processes. η- and delay bisimulation equivalence are two incomparable finer notions whereas branching bisimulation equivalence is the finest of all.

In a certain sense, the usual notion of observation equivalence does not preserve the branching structure of a process. For instance, the processes $a(\tau b + c)$ and $a(\tau b + c) + ab$ are observation equivalent. However, in the first term, in each computation the choice between b and c is made after the a-step, whereas the second term has a computation in which b is already chosen when the a-step occurs. For this reason, one may wonder whether or not we should accept the so-called third τ-law—$a(\tau x + y) = a(\tau x + y) + ax$—that is responsible for the former equivalence. Similarly, the processes $\tau a + b$ and $\tau a + a + b$ are observation equivalent. However, only in the first term every computation in which a occurs passes through a state where a did not yet happen but the possibility to do b instead is already discarded. Hence, we argue that also the second τ-law—$\tau x = \tau x + x$—(responsible for the latter equivalence) does not respect branching time.

These examples show us that while preserving observation equivalence, we can introduce new paths in a graph that were not there before. To be precise: the *traces* are the same, but the sequences of intermediate nodes are different (modulo observation equivalence), since in the definition of observation equivalence there is no restriction whatsoever on the nature of the nodes that are passed through during the execution of a sequence of τ-steps, preceding or following corresponding atomic actions. This is the key point in our definition of branching bisimulation equivalence: in two bisimilar processes every computation in the one process corresponds to a computation in the other, in such a way that all intermediate states of these computations correspond as well, due to the bisimulation relation. It turns out that it can be defined by a small change in the definition of observation equivalence.

The fact that observation equivalence can be too coarse in its identifications is illustrated even more strongly by the problems that it may cause in practical applications and analysis. As an example, it can be shown (cf. Graf and Sifakis

[1987]) that there is no modal logic with eventually operator $\blacklozenge$ which is adequate for observation equivalence. Here $p \models \blacklozenge \phi$ means that all paths from process p will sooner or later reach a state where ϕ holds. Indeed, suppose that such a logic *would* exist, then this means that two processes are observation equivalent iff they satisfy the same modal formulas. For instance, there exists a formula ϕ such that: $(\tau b + c) \models \phi$ and $b \not\models \phi$ since obviously these processes are not observation equivalent. However, from $(\tau b + c) \models \phi$ it follows that we have $a(\tau b + c) \models \blacklozenge \phi$ whereas from $b \not\models \phi$ we find $a(\tau b + c) + ab \not\models \blacklozenge \phi$ although both processes are observation equivalent. Obviously, this inconsistency is due to the third τ-law. Similarly there must be a formula ψ such that $a \models \psi$ and $\tau a + a + ba \not\models \psi$. Thus, $\tau a + ba \models \blacklozenge \psi$, whereas $\tau a + a + ba \not\models \blacklozenge \psi$, although both processes are observation equivalent. This time the inconsistency is due to the second τ-law.

A paper by Jonsson and Parrow [1993] on deciding bisimulation equivalence shows a different kind of struggle with the third τ-law. In this paper, infinite data flow is turned into a finite state representation by considering symbolic transitions. This provides us with a method to decide on the equivalence of infinite data flow programs. It turns out to work easily for strong equivalence, but for observation equivalence there is no straightforward generalization of the former results and a less intuitive transition system is needed to fix this problem. It is easy to see that using branching bisimulation would serve as a key to a more natural solution of this problem.

Having at least four options for the defintion of bisimulation congruence involving τ-steps, in any particular application it becomes important to have a clear intuition about which kind of abstraction is preferable. In an important class of problems, one can prove, however, that all four notions of bisimulation yield the same equivalence. In particular, this is the case if one of the two bisimulating graphs does not have any τ-steps. It is interesting to observe that, as far as we know, all case studies on protocol verification performed so far fit into this class of problems; hence, all of their proofs that have been given in the setting of observation equivalence still hold in branching bisimulation semantics.

2. *Branching and Abstraction*

In this section, we define the semantic equivalences that we want to discuss on a domain of *process graphs*. Since we focus on branching and abstraction, we have chosen to abstain from a proper modeling of divergence, concurrency, real-time behavior and stochastic aspects of processes. Moreover, we will disregard the nature of the actions that our processes may perform: they will be modeled as uninterpreted symbols $a, b, c, \ldots$ from a given set Act. We have chosen process graphs (or labeled transition systems) to represent processes, since they clearly visualize the aspects of the modeled systems' behavior we are interested in. The nodes in our graphs (or states in our transition systems) remain anonymous. A common alternative is to use closed expressions in a system description language like CCS or ACP as nodes in process graphs, but here we prefer to separate the semantic issues from the treatment of a particular language. In the next section, however, we will give an interpretation of a subset of CCS in the graph model and discuss the algebraic aspects of our equivalences.

Definition 2.1. A *process graph* is a connected, rooted, edge-labeled and directed graph.

In an edge-labeled directed graph, edges go from one node to another (or the same) node and are labeled with elements from a certain set Act. One can have more than one edge between two nodes as long as they carry different labels. A rooted graph has one special node, which is indicated as the root node. We require process graphs to be connected: they need not be finite, but one must be able to reach every node from the root node by following a finite path. If r and s are nodes in a graph, then $r \to^a s$ denotes an edge from r to s with label a or it will be used as a *proposition* saying that such an edge exists. Process graphs represent concurrent systems in the following way: the elements of Act are *actions* a system may perform; the nodes of a process graph represent the states of a concurrent system; the root is the initial state and if $r \to^a s$, then the system can evolve from state r to state s by performing an action a. The domain of process graphs will be denoted by $\mathbf{G}$.

On $\mathbf{G}$, we consider the notion of *bisimulation equivalence*, which originally was due to Park [1981] and used in Milner [1983; 1985; 1989] and in a different formulation already in Milner [1980]. On the domain of process graphs, a bisimulation usually is defined as a relation $R \subseteq \text{nodes}(g) \times \text{nodes}(h)$ on the nodes of graphs g and h satisfying:

(i) The roots of g and h are related by R;
(ii) If $R(r, s)$ and $r \to^a r'$, then there is a node s' such that $s \to^a s'$ and $R(r', s')$;
(iii) If $R(r, s)$ and $s \to^a s'$, then there is a node r' such that $r \to^a r'$ and $R(r', s')$.

Equivalently—as is done in this paper—one can obtain bisimulation equivalence from a *symmetric* relation R between nodes of g and h, only satisfying (i) and (ii). Such a symmetric relation can be defined as a relation $R \subseteq \text{nodes}(g) \times \text{nodes}(h) \cup \text{nodes}(h) \times \text{nodes}(g)$ such that $R(r, s) \Leftrightarrow R(s, r)$, or, alternatively, as a set of unordered pairs of nodes $R \subseteq \{\{r, s\}: r \in \text{nodes}(g), s \in \text{nodes}(h)\}$. In the latter case $R(r, s)$ abbreviates $\{r, s\} \in R$. Note that a symmetric relation satisfying (i) and (ii) also satisfies (iii), and therefore is a bisimulation as defined above. Furthermore, if R is a bisimulation between two graphs in the sense defined above, then $R \cup R^{-1}$ is a symmetric bisimulation between these graphs. Hence, the restriction to symmetric relations does not cause any loss of generality.

Definition 2.2. Two graphs g and h in $\mathbf{G}$ are *bisimilar*—notation: $\underline{\leftrightarrow}\, h$—if there exists a symmetric relation R between the nodes of g and h (called a *bisimulation*) such that:

(i) The roots of g and h are related by R;
(ii) If $R(r, s)$ and $r \to^a r'$, then there is a node s' such that $s \to^a s'$ and $R(r', s')$.

Bisimilarity turns out to be an equivalence relation on $\mathbf{G}$, which is called *bisimulation equivalence*. Depending on the context we will sometimes use Milner's terminology and refer to bisimulation equivalence as *strong equivalence* or *strong congruence*.

Now let us postulate the existence of a special action $\tau \in \mathrm{Act}$, that represents an unobservable, internal move of a process. We write $r \Rightarrow s$ for a path from r to s consisting of an arbitrary number (≥ 0) of τ-steps.

The definition of strong congruence was the starting point of Milner [1980] when he considered abstraction in CCS. Having in mind that τ-steps are not observable, he suggested to simply require that for g and h to be equivalent, (i) every possible a-step ($a \neq \tau$) in the one graph should correspond with an a-step in the other (as for usual bisimulation equivalence), apart from some arbitrary long sequences of τ-steps that are allowed to precede or follow, and (ii) every τ-step should correspond to an arbitrary long (≥ 0) τ-sequence. This way he obtained his notion of *observation equivalence* (cf. Milner [1980; 1985; 1989])—or *weak bisimulation equivalence*—which can be defined as follows:

Definition 2.3. Two graphs g and h are *weakly bisimilar*—notation: $g \leftrightarrow_{\mathrm{w}} h$ —if there exists a symmetric relation R (called a *weak bisimulation*) between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(r, s)$ and $r \to^{a} r'$, then either $a = \tau$ and $R(r', s)$, or there exists a path $s \Rightarrow s_1 \to^{a} s_2 \Rightarrow s'$ such that $R(r', s')$.

Again, $\leftrightarrow_{\mathrm{w}}$ is an equivalence on $\mathbf{G}$ which is called *weak* (*bisimulation*) *equivalence*, also known as *observation equivalence* or *τ-bisimulation equivalence*. For $\sigma = a_1 a_2 \cdots a_n$ a (possibly empty) sequence of visible actions let $p \Rightarrow^{\sigma} q$ denote $p \Rightarrow \to^{a_1} \Rightarrow \to^{a_2} \Rightarrow \cdots \Rightarrow \to^{a_n} \Rightarrow q$, that is, a path from p to q passing through a sequence of actions that reduces to σ after leaving out the internal ones. Then a weak bisimulation can be equivalently defined as a symmetric relation between the nodes of graphs g and h, such that:

(i) The roots are related by R;
(ii) If $R(r, s)$ and $r \Rightarrow^{\sigma} r'$, then there exists a path $s \Rightarrow^{\sigma} s'$ such that $R(r', s')$.

Although less useful for purposes of verification, in this (original) form the weak bisimulation shows a clear parallel with the τ-less version.

Still, to some extent, the notion of weak bisimulation cannot be regarded as the natural generalization of ordinary bisimulation to an abstract setting with hidden steps. The reason for this is that an important feature of a bisimulation is missing for weak bisimulation, namely the property that any computation in the one process corresponds to a computation in the other, in such a way that all intermediate states of these computations correspond as well, due to the bisimulation relation. When Hennessy and Milner [1980; 1985] introduced the first version of observation equivalence, they also insisted on relating the intermediate states of computations, as they tell us: "...any satisfactory comparison of the behavior of concurrent programs must take into account their intermediate states as they progress through a computation, because differing intermediate states can be exploited in different program contexts to produce different overall behavior..." and: "If we consider a computation as a sequence of experiments (or communications), then the above remarks show that the intermediate states are compared. In fact, if p is to be equivalent to q, there must be a strong relationship between their respective intermediate

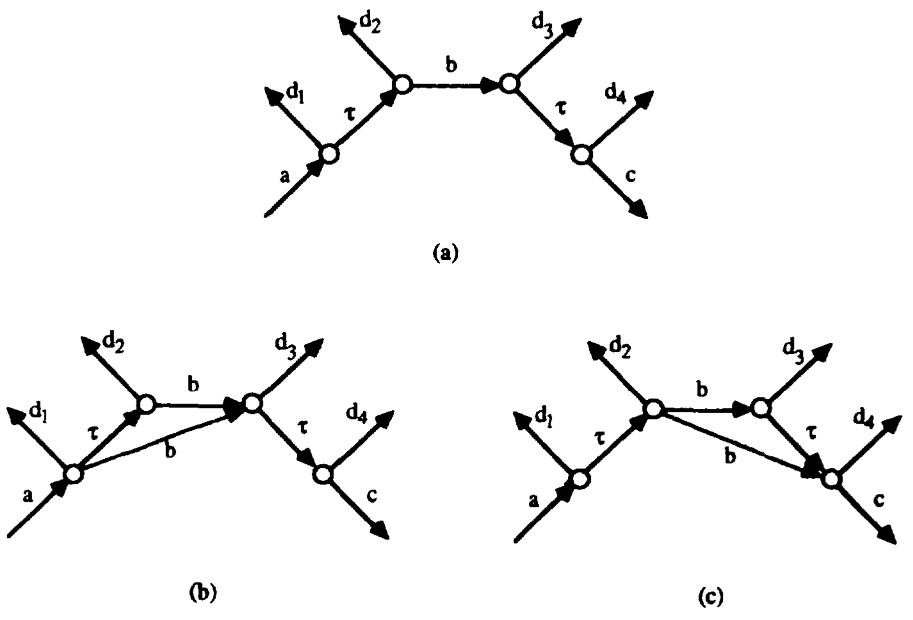

FIG. 1. Observation equivalence.

states. At each intermediate stage in the computations, the respective "potentials" must also be the same". However, in Milner's observation equivalence, when satisfying the second requirement of Definition 2.3 one may execute arbitrary many τ-steps in a graph without worrying about the status of the nodes that are passed through in the meantime.[1] As an illustration, in Figure 1 we consider a path $a\tau b\tau c$ with outgoing edges $d_1, \ldots, d_4$, and it follows easily that all three graphs are observation equivalent. Note that an extra b-edge has been added in (b) and (c) without disturbing equivalence. However, in both (b) and (c) a new computation path is introduced—in which the outgoing edge d_2 (or d_3, respectively) is missing—and such a path did not occur in (a). Or—to put it differently—in the path introduced in (b) the options d_1 and d_2 are discarded simultaneously, whereas in (a) it corresponds to a path containing a state where the option d_1 is already discarded but d_2 is still possible. Also in the path introduced in (c) the choice not to perform d_3 is already made with the execution of the b-step, whereas in (a) it corresponds to a path in which this choice is made only after the b-step. Thus, we argue that observation equivalence does not preserve the branching structure of processes and hence lacks one of the main characteristics of bisimulation semantics.

Consider the following alternative definition of bisimulation in order to see how this can be overcome.

[1] These quotes were actually intended to motivate a nonfunctional semantics for reactive processes. We also note that observational equivalence *does* take the intermediate states of related processes into account, although not on the level of matching computations. Here we merely take the same ideas that motivated Hennessy and Milner to an extreme. In the conclusion we will show how the different intermediate states of matching computations of observationally equivalent processes can be exploited by a program context to produce different overall behavior.

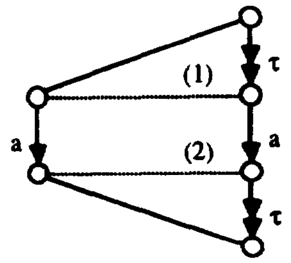

FIG. 2. Bisimulations with τ.

Definition 2.4. Two graphs g and h are *branching bisimilar*—notation: $g \leftrightarrow_b h$—if there exists a symmetric relation R (called a *branching bisimulation*) between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(r,s)$ and $r \rightarrow^a r'$, then either $a = \tau$ and $R(r',s)$, or there exists a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r,s_1)$, $R(r',s_2)$ and $R(r',s')$.

In a picture, the difference between branching and weak bisimulation can be characterized as shown in Figure 2.

The double arrow corresponds to the symbol $\Rightarrow$. Ordinary weak bisimulation (Definition 2.3) says that every a-step $r \rightarrow^a r'$ corresponds with a path $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ and so we obtain Figure 2 *without* the lines marked with (1) and (2). Branching bisimulation moreover requires relations between r and s_1 and between r' and s_2 and thus we obtain Figure 2 *with* (1) and (2). Note that if $g \leftrightarrow_b h$ then there exists a *largest* branching bisimulation between g and h, since the set of branching bisimulations is closed under arbitrary union.

Obviously, branching bisimilarity more strongly preserves the branching structure of a graph since the starting and endnodes of the τ-paths $s \Rightarrow s_1$ and $s_2 \Rightarrow s$ are related to the same nodes. Observe that in Figure 1 there are no branching bisimulations between any of the graphs (a), (b) and (c). In particular, adding extra edges as in (b) and (c) no longer preserves branching bisimilarity. Equivalently, we could have strengthened Definition 2.3 (ii) by requiring *all* intermediate nodes in $s \Rightarrow s_1$ and $s_2 \Rightarrow s$ to be related with r and r' respectively. In fact, this would yield the notion we really want to define. That Definition 2.4 yields the same relation can be seen by use of the following lemma:

LEMMA 2.5 (STUTTERING LEMMA). *Let R be the largest branching bisimulation between g and h. If $r \rightarrow^\tau r_1 \rightarrow^\tau \cdots \rightarrow^\tau r_m \rightarrow^\tau r'$ $(m \geq 0)$ is a path such that $R(r,s)$ and $R(r',s)$ then $\forall_{1 \leq i \leq m}$: $R(r_i, s)$.*

PROOF. First we prove Lemma 2.5 for a slightly different kind of bisimulation, defined as follows:

Definition 2.6. A *semi-branching bisimulation* between two graphs g and h is a symmetric relation R between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(v,w)$ and $v \rightarrow^a v'$ then either
  (a) $a = \tau$ and there exists a path $w \Rightarrow w'$ such that $R(v,w')$ and $R(v',w')$, or:
  (b) there exists a path $w \Rightarrow w_1 \rightarrow^a w_2 \Rightarrow w'$ such that $R(v,w_1)$, $R(v',w_2)$, and $R(v',w')$.

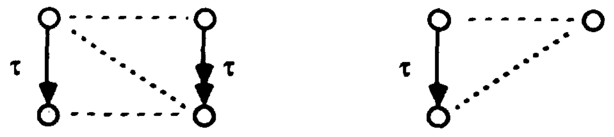

FIG. 3. Semibranching (left) and branching bisimulation.

The difference with branching bisimulation is in case (a), which can be illustrated by Figure 3.

Now let (*) denote the property, mentioned in the lemma. Observe that (a) any branching bisimulation is a semibranching bisimulation and (b) any semibranching bisimulation satisfying (*) is a branching bisimulation.

CLAIM 2.7. *The largest semibranching bisimulation between g and h satisfies* (*).

Let R be the largest semibranching bisimulation between g and h, let s be a node and let $r \to^{\tau} r_1 \to^{\tau} \cdots \to^{\tau} r_m \to^{\tau} r'$ $(m \geq 0)$ be a path such that $R(r, s)$ and $R(r', s)$. Then we prove that $R' = R \cup \{(r_i, s): 1 \leq i \leq m\}$ is a semibranching bisimulation. We check the conditions:

(i) The root nodes of g and h are related by R', since they are related by R.
(ii) Suppose $R'(v, w)$ and $v \to^{a} v'$. If $R(v, w)$, then it follows that either (a) $a = \tau$ and there exists a path $w \Rightarrow w'$ such that $R(v, w')$ and $R(v', w')$, or (b) there exists a path $w \Rightarrow w_1 \to^{a} w_2 \Rightarrow w'$ such that $R(v, w_1)$, $R(v', w_2)$, and $R(v', w')$. Hence, from $R \subseteq R'$, we find that R' satisfies the requirements in the definition above.

So assume *not* $R(v, w)$, then we find that either (1) $v = s$ and $w = r_i$ or (2) $v = r_i$ and $w = s$.

(1) If $s \to^{a} s'$, then it follows from $R(r', s)$ that either: $a = \tau$ and there is a path $r' \Rightarrow r''$ such that $R(r'', s)$ and $R(r'', s')$. Hence, there is a path $r_i \Rightarrow r' \Rightarrow r''$ such that $R'(r'', s)$ and $R'(r'', s')$ as required.
or: there is a path $r' \Rightarrow t_1 \to^{a} t_2 \Rightarrow r''$ such that $R(t_1, s)$, $R(t_2, s')$, and $R(r'', s')$ and hence $r_i \Rightarrow r' \Rightarrow t_1 \to^{a} t_2 \Rightarrow r''$ with $R'(t_1, s)$, $R'(t_2, s')$, and $R'(r'', s')$.
(2) If $r_i \to^{a} r''$, then $r \to^{\tau} r_1 \to^{\tau} \cdots \to^{\tau} r_i \to^{a} r''$, and since $R(r, s)$, we find that there exists a sequence $s \Rightarrow s_1 \Rightarrow \cdots \Rightarrow s_i$ such that $R(r_1, s_1), \ldots, R(r_i, s_i)$. It follows from $R(r_i, s_i)$ that either: $a = \tau$ and there exists a path $s_i \Rightarrow s'$ such that $R(r_i, s')$ and $R(r'', s')$. Hence, $s \Rightarrow s'$ with $R'(r_i, s')$ and $R'(r'', s')$ as required.
or: there exists a path $s_i \Rightarrow t_1 \to^{a} t_2 \Rightarrow s''$ such that $R(r_i, t_1)$, $R(r'', t_2)$ and $R(r'', s'')$, and hence $s \Rightarrow s_i \Rightarrow t_1 \to^{a} t_2 \Rightarrow s''$ with $R'(r_i, t_1)$, $R'(r'', t_2)$ and $R'(r'', s'')$.

This proves that R' is a semibranching bisimulation. Since R is the largest, we find $R = R'$.

So we proved the claim. Finally, conclude that the largest semi-branching bisimulation is equal to the largest branching bisimulation, and thus we proved the lemma. □

The stuttering lemma will play a crucial role in some of the results we will present later. An alternative proof can be found in De Nicola et al. [1990].

It follows from Figure 2 that we can find two more kinds of bisimulation with τ, since we can leave out (1) while still having (2) and vice versa. Consider the following two definitions:

Definition 2.8. Two graphs g and h are *η-bisimilar*—notation $g \underline{\leftrightarrow}_\eta h$—if there exists a symmetric relation R (called an *η-bisimulation*) between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(r,s)$ and $r \to^a r'$, then either $a = \tau$ and $R(r',s)$, or there exists a path $s \Rightarrow s_1 \to^a s_2 \Rightarrow s'$ such that $R(r,s_1)$ and $R(r',s')$.

Definition 2.9. Two graphs g and h are *delay bisimilar*—notation $g \underline{\leftrightarrow}_d h$—if there exists a symmetric relation R (called a *delay bisimulation*) between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(r,s)$ and $r \to^a r'$, then either $a = \tau$ and $R(r',s)$, or there exists a path $s \Rightarrow s_1 \to^a s_2 \Rightarrow s'$ such that $R(r',s_2)$ and $R(r',s')$.

Notice the subtle differences between both definitions (and Definition 2.4). In Definition 2.8 the notion of η-bisimulation corresponds to Figure 2 without the relation (2) but with (1). Similarly, with delay bisimulation we have (2) but not (1). It is easy to see that in the definition of both branching and delay bisimulation the existence requirement of a node s' such that $s_2 \Rightarrow s'$ and $R(r',s')$ is redundant.

From the definitions, we find immediately that $g \underline{\leftrightarrow}_b h \Rightarrow g \underline{\leftrightarrow}_\eta h \Rightarrow g \underline{\leftrightarrow}_w h$ and similarly $g \underline{\leftrightarrow}_b h \Rightarrow g \underline{\leftrightarrow}_d h \Rightarrow g \underline{\leftrightarrow}_w h$. Observe that in Figure 1 we find an η-bisimulation between (a) and (c) and a delay bisimulation between (a) and (b). Conversely, there is no η-bisimulation between (a) and (b) and no delay bisimulation between (a) and (c), so all implications are strict.

The notion of η-bisimulation was first introduced by Baeten and Van Glabbeek [1987] as a finer version of observation equivalence. A variant of delay bisimulation—only differing in the treatment of divergence—first appeared in Milner [1981], also under the name observational equivalence.

PROPOSITION 2.10. *For $* \in \{w, \eta, d\}$, $\underline{\leftrightarrow}_*$ is an equivalence relation on $\mathbb{G}$.*

PROOF. $\underline{\leftrightarrow}_*$ is reflexive since the identity relation is a $*$-bisimulation between any graph and itself, and it is symmetric by definition. Furthermore let R be a $*$-bisimulation between g and g', and S a $*$-bisimulation between g' and g''. Their composition $R \circ S$ is defined by

$$R \circ S(r,r''): \Leftrightarrow \text{ for some } r' \text{ in } g'\text{: } R(r,r') \text{ and } S(r',r'').$$

Now one can easily prove that $R \circ S \cup S \circ R$ is a $*$-bisimulation between g and g'', so $\underline{\leftrightarrow}_*$ is transitive. □

In Van Glabbeek and Weijland [1991] we claimed that the proof above would also work for $* = b$, showing that branching bisimilarity is an equivalence. However, Basten [1996] showed that this is not the case, since $R \circ S \cup S \circ R$ need not be a branching bisimulation.

PROPOSITION 2.11. *Branching bisimilarity is an equivalence relation on* $\mathbf{G}$.

PROOF. Reflexivity and symmetry proceed as before. Let R be a branching bisimulation between g and g', and S a branching bisimulation between g' and g''. Now one can easily prove that $R \circ S \cup S \circ R$ is a semi-branching bisimulation (as in Definition 2.6) between g and g'', which implies that $g \leftrightarrow_b g''$. Thus, $\leftrightarrow_*$ is transitive. This proof is elaborated in detail in Basten [1996].

For an alternative transitivity proof assume $g \leftrightarrow_b g' \leftrightarrow_b g''$. Then, there exists branching bisimulations R between g and g' and S between g' and g'' satisfying the property of Lemma 2.5. Now $R \circ S \cup S \circ R$ is a branching bisimulation between g and g''.

A third way to obtain Proposition 2.11 is as an immediate consequence of Theorem 4.10. □

We therefore refer to $\leftrightarrow_b$ as *branching bisimulation equivalence* or *branching equivalence* for short.

3. *From Equivalence to Congruence*

In this section, we will turn our graph domain $\mathbf{G}$ into an algebra, by defining some operations on them. This will enable us to give equationial characterizations of the equivalences studied in the previous section. Here we restrain ourselves to the operators inaction, prefixing and alternative composition of CCS (cf. Milner [1980]). In the technical report version of this paper (Van Glabbeek and Weijland [1991]) we also consider the sequential composition operator of ACP (cf. Bergstra and Klop [1984]) and address the issue of deadlock versus successful termination. We will not consider parallel composition, restriction (or encapsulation), hiding and relabeling. However, we claim that these can be added without problem (see also Baeten & Weijland [1990]).

For sake of convenience, in this section we will only consider *root-unwound* process graphs, that is, process graphs with no incoming edges at the root. Since each bisimulation equivalence class of process graphs contains a root-unwound graph, this does not cause any loss of generality. The domain of root-unwound process graphs will be denoted by $\mathbf{G}^{\rho}$. Clearly $\mathbf{G}^{\rho} \subseteq \mathbf{G}$.

In order to equip $\mathbf{G}^{\rho}$ with some structure, we introduce a constant 0 for inaction, a binary infix written operator + for alternative composition, and unary operators $a.$ for prefixing for every element $a \in \text{Act}$. As we only are interested in process graphs up to isomorphism (or even strong bisimulation), we may bijectively rename their nodes when it suits us in the definition of an operator.

Definition 3.1. The constant 0 and the operators + and $a.$ are defined on $\mathbf{G}^{\rho}$ as follows:

(i) The constant 0 is interpreted as the trivial graph, having one node (the root) and no edges;
(ii) $(g + h)$ can be constructed by identifying the root nodes of disjoint copies of g and h;
(iii) $(a.g)$ is constructed from g by adding a new node which will be the root of $a.g$, and a new a-labeled edge from the root of $a.g$ to the root of g.

[Formally, identifying is defined as dividing out the equivalence relation 'to be identified' on the disjoint union of the nodes of g and h. The convention applies that any pointer (such as a name) to a node of g or h is also used to refer to its equivalence class, this being a node of $g + h$. In particular, if "R" is the name of a relation between the nodes of g and g', then "R" is also the name of a relation between the nodes $g + h$ and g', or $g + h$ and $g' + h'$. Which of these relations is meant by a given occurrence of "R" is determined by the context.]

We often leave out brackets, assuming that $+$ will be the weakest binding operator symbol, and abbreviate $a.p$ by ap and $a.0$ by a. Moreover, we overload the notation Act to denote the set of prefix operators $\{a.: a \in \text{Act}\}$. Milner [1980] proved that the operators from Act and $+$ all are well-defined on $\mathbf{G}^p \underline{\leftrightarrow}$, even after dividing out bisimulation equivalence. This follows from the following theorem, the proof of which is straightforward and omitted.

THEOREM 3.2 *Bisimulation equivalence is a congruence with respect to the operators from Act and* $+$.

Hence the structure $(\mathbf{G}^p \underline{\leftrightarrow}, 0, +, Act)$ is a well-defined algebra. Considering its first order equational theory one finds the axioms shown in Table I.

Now let us say that a theory Γ is a *complete axiomatization* of a model $\mathbf{M}$ if for every pair of closed terms p and q we have: $\Gamma \vdash p = q$ if and only if $\mathbf{M} \models p = q$. This definition deviates from the standard one, since usually also open terms are considered.

Let us call the theory from Table I *Basic CCS*, and write BCCS = A0–A3. Then, the following theorem is due to Hennessy and Milner [1980] and Milner [1980].

THEOREM 3.3. *BCCS is a complete axiomatization of* $(\mathbf{G}^p/\underline{\leftrightarrow}, 0, +, Act)$.

In the same way, one may wish to find axiomatizations for algebras resulting from dividing out the other equivalences of Section 2. However, as it turns out, these equivalences are not congruences with respect to the operator $+$. In the case of observation equivalence, this problem was solved by Milner [1980] by simply taking the closure of $\underline{\leftrightarrow}_w$ with respect to all contexts in CCS, thereby obtaining *observation congruence*. Similarly in Hennessy and Milner [1980], *observational congruence* was defined as the CCS-closure of their earlier variant of observational equivalence (the difference between the two variants is discussed in the historical note in Van Glabbeek and Weijland [1991], the technical report version of this paper) and this congruence coincides with the one of Milner [1980]. Bergstra and Klop [1985] formulated an additional condition, yielding an immediate definition of observation congruence by means of bisimulation relations.

Definition 3.4 (*root condition*). A relation R between nodes of process graphs is called *rooted* if root nodes are related with root nodes only.

Observe that whenever two root unwound process graphs are bisimulation equivalent in the sense of Definition 2.2, there exists a rooted bisimulation between them, but this is not necessarily the case for the relations defined in Definitions 2.3, 2.4, 2.8, and 2.9. For any two process graphs g and h in $\mathbf{G}^p$ and $* \in \{w, b, \eta, d\}$ we write R: $g \underline{\leftrightarrow}_{r*} h$ if R is a rooted $*$-bisimulation between g and h, and $g \underline{\leftrightarrow}_{r*} h$ if such a relation exists.

TABLE I. BASIC CCS

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$x + 0 = x$	A0

THEOREM 3.5. *For* $* \in \{w, b, \eta, d\}$, $\underline{\leftrightarrow}_r*$ *is a consequence on* G^p *with respect to* + *and Act.*

PROOF. We prove Theorem 3.5 for $\underline{\leftrightarrow}_{rb}$. The other proofs proceed in the same way.

As in preposition 2.11, one shows that $\underline{\leftrightarrow}_{rb}$ is an eqivalence relation. So it is left to prove that $\underline{\leftrightarrow}_{rb}$ respects the operators.

$\pm$: Suppose that R: $g \underline{\leftrightarrow}_{rb} g'$ and S: $h \underline{\leftrightarrow}_{rb} h'$.

We prove that $(R \cup S)$: $(g + h) \underline{\leftrightarrow}_{rb} (g' + h')$.

(i) Obviously, the roots of $(g + h)$ and $(g' + h')$ are related.

(ii) Assume that in $(g + h)$ we have an edge $r \to^a r'$ and suppose we have $(R \cup S)(r, s)$, then from the construction of $(g + h)$, it follows that this edge either originates from g or from h; let us say it is g. It follows from $(R \cup S)(r, s)$ that either $R(r, s)$ or $S(r, s)$, so we have two distinct cases:

Firstly, suppose that $R(r, s)$. Then, either $a = \tau$ and $R(r', s)$—hence, $(R \cup S)(r', s)$ and $(R \cup S)$ satisfies Definition 2.4—or there exists a path $s \Rightarrow s_1 \to^a s_2 \Rightarrow s'$ in g' such that $R(r, s_1)$, $R(r', s_2)$, and $R(r', s')$. Obviously, we can find the same path in $(g' + h')$ and we have that $(R \cup S)(r, s_1), (R \cup S)(r', s_2)$ and $(R \cup S)(r', s')$ as required.

Secondly, suppose that we do *not* have $R(r, s)$. Then, we have $S(r, s)$, and since we assumed that the edge $r \to^a r'$ came from g, we find that r has to be the (joint) root node of g and h. However, in S root nodes are related with root nodes only (the root condition), and so s must be the joint root node of g' and h'. Hence, $R(r, s)$, which is a contradiction.

(iii) Obviously, the root nodes of $(g + h)$ and $(g' + h')$ are uniquely related by $(R \cup S)$.

a: It remains to prove that $\underline{\leftrightarrow}_{rb}$ respects the operators in Act. Suppose that R: $g \underline{\leftrightarrow}_{rb} g'$ and p, p' are the root nodes of $a.g$ and $a.g'$. Put $R^* := R \cup \{p, p'\}$. Then, we prove R^*:$(a.g) \underline{\leftrightarrow}_{rb} (a.g')$.

(i) Clearly, the roots of both graphs are related by R^*.

(ii) Assume that in $(a.g)$ we have an edge $r \to^b r'$ and suppose we have $R^*(r, s)$, then from the construction of $(a.g)$ it follows that either $r = p$ or this edge originates from g.

If $r = p$, then, by the definition of R^*, we have $s = p'$. Furthermore, $b = a$ and r' is the root node of g, and by the construction of prefixing we find that in g' there exists an edge $s \to^a s'$ to the root node s' of g'. Since R is a branching bisimulation, we find $R(r', s')$ and hence $R^*(r', s')$.

If $r \to^b r'$ originates from g, then it follows from the definition of R^* that $R(r, s)$, from which the reqirement follows immediately.

(iii) Clearly, the root nodes are uniquely related by R^*. □

THEOREM 3.6. *Provided that there exists at least one action $a \in Act$ with $a \neq \tau$, $\underline{\leftrightarrow}_{r*}$ is the coarsest congruence on $\mathbf{G}^p$ with respect to $+$ that is contained in $\underline{\leftrightarrow}_*$, for $* \in \{w, b, \eta, d\}$. Hence $\underline{\leftrightarrow}_{rw}$ coincides with observation congruence.*

PROOF. The idea for this proof is due to J. W. Klop (personal communication). Let g and $h \in \mathbf{G}^p$ and suppose $g + k \underline{\leftrightarrow}_* h + k$ for any graph $k \in \mathbf{G}^p$. Suppose there is an action $a \in$ Act $(a \neq \tau)$ that does not occur in g and h. Then $g + a \underline{\leftrightarrow}_* h + a$. Let R be a $*$-bisimulation between $g + a$ and $h + a$, then R must be rooted. Therefore, the restriction of R to the nodes of g and h is a rooted $*$-bisimulation between g and h.

If no 'fresh atom' $a \in$ Act can be found, a variant of this method still works. First, note that for each infinite cardinal κ there are at least κ $*$-bisimulation equivalence classes of graphs with less than κ nodes. (Choose an action $a \in$ Act $(a \neq \tau)$ and define for each ordinal λ the graphs g_λ as follows: $g_0 = 0$, $g_{\lambda+1} = g_\lambda + ag_\lambda$ and for λ a limit ordinal g_λ is constructed from all graphs g_μ for $\mu < \lambda$ by identifying their roots. Then, with transfinite induction, it follows that no two different g_λ's are $*$-bisimilar. Furthermore, for infinite λ, the cardinality of the nodes of g_λ is the cardinality of λ.) Thus, for any two graphs g and h there must be a graph $k \in \mathbf{G}^p$ with the same cardinality such that k is not bisimilar with any subgraph corresponding with a node in g or h. Now take a $*$-bisimulation between $g + \tau k$ and $h + \tau k$. □

The equivalence relations $\underline{\leftrightarrow}_{r*}$ are called *rooted $*$-bisimulation equivalence* or *$*$-bisimulation congruence*. As a consequence of Theorem 3.5, we find that all structures $(G^p/\underline{\leftrightarrow}_{r*}, 0, +, \text{Act})$ are well-defined algebras.

Theorem 3.6 says that (if Act-$\{\tau\}$ is nonempty) both methods to define weak bisimulation congruence (by closure under context or using the root condition) yield the same result. In this respect η-, delay and branching bisimulation behave exactly the same. However, each weak bisimulation equivalence class consists of at most two weak bisimulation congruence classes (this follows from Exercise 7.6 of Hennessy in Milner [1980]), as is the case for delay bisimulation, whereas η- and branching bisimulation equivalence classes may contain many congruence classes. Nevertheless, for all four bisimulations there exists a close relationship between rooted and non-rooted bisimulation, since the root condition (Definition 3.4) only works on the root nodes:

THEOREM 3.7. *For all root unwound graphs g and h and $* \in \{w, b, \eta, d\}$ we have:*

$$g \underline{\leftrightarrow}_* h \quad \textit{if and only if} \quad \tau.g \underline{\leftrightarrow}_{r*} \tau.h.$$

PROOF. If R is a $*$-bisimulation between g and h and r, s are the roots of $\tau.g$ and $\tau.h$, then $R \cup \{r, s\}$ is a rooted $*$-bisimulation between $\tau.g$ and $\tau.h$. On the other hand, if R is a rooted $*$-bisimulation between $\tau.g$ and $\tau.h$, then the roots of g and h are related by R, so R restricted to the nodes of g and h is a $*$-bisimulation between g and h. □

This theorem provides us with a tool to decide upon $*$-bisimulation equivalence, using the axiom systems of $*$-bisimulation congruence that will be provided in Section 5.

In defining rooted bisimulations we restricted ourselves to root-unwound process graphs. The reason for doing so is that the definitions given do not work for graphs with incoming edges in the root. As remarked before already, this does not cause any loss of generality. It is straightforward to define an operator ρ for root-unwinding, such that for any graph g, $\rho(g)$ is root-unwound and strongly bisimilar with g (and moreover if g is root-unwound already, $\rho(g)$ and g are isomorphic). Now the definitions of rooted bisimulation can be extended to all process graphs by letting $g \leftrightarrow_{r*} h$ iff $\rho(g) \leftrightarrow_{r*} \rho(h)$. In case we do not restrict to root-unwound process graphs, the definition of the operator $+$ on process graphs has to be upgraded as well. An easy way to do so is by defining $g + h$ as $\rho(g) + \rho(h)$.

A rooted branching bisimulation between two graphs g and h can alternatively be defined as a branching bisimulation R such that, for a visible or invisible:

(iii) If $\text{root}(g) \rightarrow^{a} r$, then there is an edge $\text{root}(h) \rightarrow^{a} s$ with $R(r, s)$.
(iv) If $\text{root}(h) \rightarrow^{a} s$, then there is an edge $\text{root}(g) \rightarrow^{a} r$ with $R(r, s)$.

This means that a rooted branching bisimulation behaves like a strong bisimulation in the first step, and like a branching bisimulation further on. It is easy to see that for root-unwound graphs this definition agrees with Definition 3.4. However, it has the advantage that it can directly be used for graphs that are not root-unwound. Also delay, η- and weak bisimulation congruence can be characterized in a similar fashion (cf. Milner [1989] and Baeten and Van Glabbeek [1987]), but giving up the analogy with a strong bisimulation in the first step. However, the characterizations of this type of the various bisimulation congruences have a rather diverse appearance, whereas the root condition gives a uniform way for turning weak, delay, η- and branching bisimulation into a congruence.

4. *Branches and Traces*

As we saw in Figure 1, while perserving observation equivalence we are able to introduce new 'paths' in a graph. To be more precise: in these new paths alternative options may branch off in places different from any in the old paths. So far, we claimed to have solved this problem by defining a new kind of bisimulation, but as of yet we still have to prove that our solution solves the problem in a fundamental way. In this section, we will establish an alternative characterization of branching bisimulation. In fact, we will show how branching bisimulation preserves the branching structure of graphs. Let us first consider ordinary bisimulation.

Definition 4.1. A *concrete trace* of a process graph is a finite sequence $(a_1, a_2, a_3, \ldots, a_k)$ of actions from Act, such that there exists a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ from the root node r_0.

Two graphs g and h are said to be *concrete trace equivalent*, notation $g \equiv_t h$, if their *concrete trace sets* (i.e., the sets of their concrete traces) are equal. It is easily checked that $\equiv_t$ is a congruence on $\mathbf{G}$ and $g \leftrightarrow h \Rightarrow g \equiv_t h$. Consequently we find that $\mathbf{G}/\equiv_t$ is a model of BCCS. Compared to bisimulation, concrete trace equivalence makes many more identifications. For example, we find that $G/\equiv_t$ satisfies the equation $a(x + y) = ax + ay$ which cannot be proved from BCCS.

The main reason for this is that a concrete trace does not provide us with information about the branching potentials in the intermediate nodes. In the following we will use *colors* at the nodes to indicate these potentials.

Definition 4.2. A *colored graph* is a process graph with colors $C \in \mathbf{C}$ as labels at the nodes.

Obviously, in a colored graph we have traces which have colors in the nodes:

Definition 4.3. A *concrete colored trace* of a colored graph g is a sequence of the form $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$ for which there exists a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ in g, starting from the root node r_0, such that r_i has color C_i.

The concrete colored traces of a node r in a graph g are the concrete colored traces of the subgraph $(g)_r$ of g that has r as its root node. This graph is obtained from g by deleting all nodes and edges that are inaccessible from r.

The question remains how to detect the color differences of the nodes, or—to put it differently—how to define the concept of 'branching potential in a node' properly. There are several ways to do this. A natural definition is the following:

Definition 4.4. A *concrete consistent coloring* of a set of graphs is a coloring of their nodes such that two nodes have the same color only if they have the same concrete colored trace set.

Obviously, the *trivial* coloring—in which every node has a different color—is consistent on any set of graphs. Note that—even apart from the choice of the colors—a set of graphs can have more than one consistent coloring. For instance, consider a set containing only an infinite graph representing a^ω or $a.a.a \cdots$ then obviously the *homogeneous* coloring—in which every node has the same color—is a consistent one, as well as the *alternating* or the trivial coloring.

Let us say two graphs g and h are *concrete colored trace equivalent*—notation: $g \equiv_{cc} h$—if for some concrete consistent coloring on $\{g, h\}$ they have the same concrete colored trace set, or equivalently, if for some concrete consistent coloring on $\{g, h\}$ the root nodes have the same color. Then, we have the following important characterization:

THEOREM 4.5. *$g \leftrightarrow h$ if and only if $g \equiv_{cc} h$.*

PROOF. $\Rightarrow$: Suppose R is the largest bisimulation relation between g and h. Let $\underline{R}$ be the transitive closure of R, then $\underline{R}$ is an equivalence relation on the set of nodes from g and h. Let $\mathbf{C}$ be the set of equivalence classes induced by $\underline{R}$ and label every node with its own equivalence class. Then this coloring is consistent on g and h.

To see this let r_0 be a node in g say, and $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$ be a concrete colored trace, which corresponds to a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} r_2 \rightarrow \cdots \rightarrow^{a_k} r_k$ starting from r_0. Now suppose for some node s_0 in h we have $R(r_0, s_0)$, then we find from Definition 2.2 that $s_0 \rightarrow^{a_1} s_1$ for some s_1 such that $R(r_1, s_1)$. Thus, r_1 and s_1 have the same color C_1. By induction, we find that s_0 has the same concrete colored trace $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$. So R preserves concrete colored trace sets; hence, so does $\underline{R}$.

Since the roots of g and h are related we find $g \equiv_{cc} h$.

$\Leftarrow$: Suppose that g and h have the same concrete colored trace sets. Then consider the relation R that relates two nodes of g and h iff they are labelled with the same color. It is easy to prove that R is a bisimulation between g and h. □

So far we did not have any notion of abstraction in the definition of colored traces, so if a colored graph has τ-labels then these are treated as if they were ordinary actions. In the following definition we find how to abstract from these τ-steps. The idea is simple: τ-steps can only be left out if they are *inert*, meaning that they are between two nodes that have the same color. Thus, it is not only that inert steps are not observable, but even more, they do not cause any change in the overall state of the machine.

Definition 4.6. A *colored trace* of a colored graph is a sequence $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$, which is obtained from a concrete colored trace of this graph by replacing all subsequences $(C, \tau, C, \tau, \ldots, \tau, C)$ by C.

Definition 4.7. A *consistent coloring* of a set of graphs is a coloring of their nodes with the property that two nodes have the same color only if they have the same colored trace set. Furthermore, such a coloring is *rooted* if no root-node has the same color as a nonroot node.

For two root-unwound graphs g and h let us write $g \equiv_c h$ if for some consistent coloring on $\{g, h\}$ they have the same colored trace set, and $g \equiv_{rc} h$ if moreover this coloring is rooted. Then we find the following characterization for (rooted) branching bisimulation:

THEOREM 4.8

(i) $g \leftrightarrow_b h$ *if and only if* $g \equiv_c h$
(ii) $g \leftrightarrow_{rb} h$ *if and only if* $g \equiv_{rc} h$.

PROOF. $\Rightarrow$: Suppose R is the largest (rooted) branching bisimulation between g and h. Let $\underline{R}$ be its transitive closure and C the set of equivalence classes induced by $\underline{R}$. Then the coloring in which every node is labeled with its own equivalence class is consistent (and rooted) on g and h.

To see this, let us write $C(r)$ for the color of the node r and assume that, for certain nodes r_0 and s_0, $R(r_0, s_0)$ and r_0 has an colored trace $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$. Then there exists a path of the form $r_0 \to^\tau u_1 \to^\tau \cdots \to^\tau u_m \to^{a_1} r_1$ $(m \geq 0)$ such that $C(r_1) = C_1$ and for all i: $C(u_i) = C(r_0) = C_0$. For every edge $u_i \to^\tau u_{i+1}$ $(0 \leq i < m, u_0 = r_0)$ there exists a path $v_i \Rightarrow v_{i+1}$ $(v_0 = s_0)$ such that $R(u_i, v_i)$, and all intermediate nodes are related to either u_i or u_{i+1} (by Lemma 2.5); hence, all v_i have the same color C_0. So we find a path $s_0 \Rightarrow v_m$ with only one color in the nodes such that $R(u_m, v_m)$.

Next, since $u_m \to^{a_1} r_1$ and $R(u_m, v_m)$ we find that either $a_1 = \tau$ and $R(r_1, v_m)$—in which case $C_1 = C_0$ in contradiction with $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$ being a colored trace—or there is a path $v_m \Rightarrow t_1 \to^{a_1} s_1$ such that $R(u_m, t_1)$ and $R(r_1, s_1)$. Again by Lemma 2.5, we find that t_1 and all the intermediate nodes in $\Rightarrow$ have the same color as v_m and so we find a colored trace (C_0, a_1, C_1) of s_0. By repeating this argument k times, we find that s_0 has a colored trace $(C_0, a_1, C_1, a_2, C_2, \ldots, a_k, C_k)$ and so R preserves colored trace sets. Thus, $\underline{R}$ induces a consistent coloring and since the roots are related we find $g \equiv_c h$. If moreover R is rooted, then so is the induced coloring.

$\Leftarrow$: Consider a (rooted) consistent coloring such that the colored trace sets of g and h are equal with respect to that coloring. Let R be the relation between nodes of g and h relating two nodes if and only if they have the same color, then it is easy to see that R is a (rooted) branching bisimulation. □

This characterization provides us with a clear intuition about what branching bisimulation actually is, since the different between *inert* steps—not changing the state of the machine—and relevant τ-steps—that behave as common actions—is visualized immediately by the (change of) colors at the nodes. It follows that branching bisimulation equivalence preserves computations together with the potentials in all intermediate states that are passed through.

Another way of looking at the coloring of a graph is the following: Since trace-equivalence is too weak to characterize branching bisimilarity, we can add more information to traces in order to distinguish between processes. Consider the following definition:

Definition 4.9. For ordinals α the *α-trace set* of a graph g is defined as follows:

(1) The α-trace set of a node r of g is the set of all γ-traces of r, for $\gamma < \alpha$.
(2) An α-trace of r is made of a sequence $(T_0, a_1, T_1, a_2, \dots, a_k, T_k)$, where a_i are actions from Act and T_i are α-trace sets such that g has a path $r_0 \rightarrow^{a_1} r_1 \rightarrow^{a_2} \cdots \rightarrow^{a_k} r_k$ and r_i has α-trace set T_i, by replacing all subsequences $(T, \tau, T, \tau, \dots, \tau, T)$ by T.
(3) The α-trace set of g is the α-trace set of its root.

Note that all 0-trace sets are empty, and the 1-trace set of g is just the set of its concrete traces from which τ's have been left out. Two graphs g and h are *α-trace equivalent*—notation $g \approx_\alpha h$—if they have the same α-trace set; they are *hypertrace equivalent*—notation $g \approx h$—if $g \approx_\alpha h$ for all ordinals α. This is clearly an equivalence relation. Note that if $\lambda < \alpha$, then $g \approx_\alpha h$ implies $g \approx_\lambda h$. From this, it immediately follows that, if $G' \subseteq G$ is a *set* of process graphs, then on G' the notion of α-trace equivalence stabilizes for some ordinal (i.e., there exists a *closure ordinal* α such that, for $g, h \in G'$, $g \approx h$ iff $g \approx_\alpha h$). In particular, if G has cardinality β, then β must be a closure ordinal. It will follow from the proof of Theorem 4.10 that the smallest ordinal with $(g \approx_\alpha h \Leftrightarrow g \approx_{\alpha+1} h)$ is a closure ordinal. Next, we prove that hypertrace equivalence coincides with colored trace equivalence:

THEOREM 4.10. *$g \approx h$ if and only if $g \equiv_c h$.*

PROOF. $\Rightarrow$: Let G' be a set of process graphs containing g, h and all their subgraphs and let α be the smallest ordinal such that, for $g', h' \in G'$, $g' \approx_\alpha h'$ iff $g' \approx_{\alpha+1} h'$. If $g' \approx_{\alpha+1} h'$, then from Definition 4.9 we find that g' and h' have the same α-traces. Consider the coloring on g and h in which every node is colored with its own α-trace set. Now a colored trace $(C_0, a_1, C_1, a_2, \dots, a_k, C_k)$ of a node r is precisely an α-trace and by definition of α we have that r and r' have the same α-trace set only if they have the same α-traces, that is, they have the same color only if they have the same colored traces. Hence, the coloring is consistent.

Now $g \approx h \Rightarrow g \approx_{\alpha+1} h \Rightarrow g$ and h have the same colored traces $\Rightarrow g \equiv_c h$.

$\Leftarrow$: Take a consistent coloring on g and h such that the roots of g and h have the same color. Then, with transfinite induction on γ, it is easy to prove that equally colored nodes have the same γ-traces for all ordinals γ. □

Hence, we find that $\approx$ coincides with $\equiv_c$, and hence with $\underline{\leftrightarrow}_b$ (Theorem 4.8). This also yields an alternative proof showing that $\underline{\leftrightarrow}_b$ is an equivalence relation. Note that compared to *readiness semantics* (cf. Olderog and Hoare [1986]), *possible-futures semantics* (cf. Rounds and Brookes [1981]) and *ready-trace semantics* (cf. Baeten et al. [1987b]) in an α-trace ($\alpha \geq 1$) we keep track of a lot more information. Apart from just all one-step exits from the endstate of a partial execution we are now able to see all traces (and higher traces) that can be chosen at every intermediate state during the execution.

The notion of hypertrace equivalence gives us an indication of the amount of extra information that is needed to turn trace equivalence into branching bisimulation equivalence. Furthermore, it provides us with an idea of how to build a consistent coloring on a set of graphs by distinguishing more and more between nodes. A construction similar to Definition 4.9 was used by Milner [1985] to characterize observation equivalence.

As a tool for further analysis we have the following proposition:

PROPOSITION 4.11. *It is possible to color the nodes of a root unwound process graph g in such a way that two nodes have the same color iff they can be related by a rooted branching autobisimulation g (relating g with itself). This coloring is rooted and consistent.*

PROOF. For every root unwound process graph g the largest rooted branching autobisimulation on g is an equivalence relation on the nodes. It follows from the proof of Theorem 4.8 that every node can be labeled with its equivalence class as a color, in order to obtain a rooted consistent coloring. □

This coloring of a graph is called its *canonical coloring*. Note that two different nodes r and s of a root unwound process graph g have the same color with respect to its canonical coloring if and only if $r, s \neq \text{root}(g)$ and $(g)_r \underline{\leftrightarrow}_b (g)_s$ (the subgraph $(g)_r$ of g with root r is defined in the beginning of this section; also remember that in the canonical coloring root nodes have colors different from those of internal nodes). In this case, we say that r and s are *rooted branching bisimilar.*

Definition 4.12. A root unwound graph is said to be in *normal form* if each node has a different color with respect to its canonical coloring and it has no τ-loops $r \to^{\tau} r$.

Next we show that each root unwound process graph is rooted branching bisimilar with exactly one normal form (up to isomorphism).

Definition 4.13. Let g be a root unwound process graph and consider its canonical coloring with color set $\mathbf{C}$. Let $N(g)$—the *normal form* of g—be the graph which can be found from g by contracting all nodes with the same color and removing τ-loops. To be precise:

(1) $N(g)$ has colors $C \in \mathbf{C}$ as its nodes;
(2) $N(g)$ has an edge $C \to^{a} C'$ ($a \neq \tau$) iff g has an edge $r \to^{a} r'$ such that $C(r) = C$ and $C(r') = C'$, where $C(r)$ denotes the color of the node r;

(3) $N(g)$ has an edge $C \rightarrow^{\tau} C'$ iff $C \neq C'$ and g has an edge $r \rightarrow^{\tau} r'$ with $C(r) = C$ and $C(r') = C'$.

PROPOSITION 4.14. *For all root unwound process graphs* g: $g \leftrightarrow_{\mathrm{rb}} N(g)$.

PROOF. Consider the canonical coloring on g, and the trivial coloring on $N(g)$ in which each node (being a color from **C**) is labeled by itself. Let R be the relation relating nodes from g and $N(g)$ iff they have the same color. Now it follows directly from the construction above that R is a rooted branching bisimulation between g and $N(g)$. □

So in every rooted branching bisimulation equivalence class of root-unwound process graphs there is a normal form. We proceed by showing that up to isomorphism there is only one.

Definition 4.15. A *graph isomorphism* is a bijective relation R between the nodes of two process graphs g and h such that:

(1) the roots of g and h are related by R;
(2) if $R(r, s)$ and $R(r', s')$, then $r \rightarrow^{a} r'$ is an edge in g iff $s \rightarrow^{a} s'$ is an edge in h ($a \in \mathrm{Act}$).

Note that a graph isomorphism is just a bijective bisimulation, or a bijective branching bisimulation for that matter. Two graphs are isomorphic—notation $g \cong h$—iff there exists an isomorphism between them. In that case g and h only differ with respect to the identity of the nodes. Note that $\cong$ is a congruence relation on process graphs.

THEOREM 4.16 (NORMAL FORM THEOREM). *Let g and h be root unwound graphs that are in normal form. Then $g \leftrightarrow_{\mathrm{rb}} h$ if and only if $g \cong h$.*

PROOF. This follows since any bisimulation R: $g \leftrightarrow_{\mathrm{rb}} h$ must be bijective:

(i) it is surjective because every node in g or h can be reached from the root; hence, by Definition 2.4, every node is related to some node in the other graph;
(ii) it is injective since every node is related with at most one other node: If two different nodes in g are related to the same node in h, then these two are also related by a branching autobisimulation on g, and so with respect to the canonical coloring they have the same color. But then by Definition 4.12 the nodes are identical, which is a contradiction. □

Theorem 4.16 says that each equivalence class in $\mathbb{G}/\leftrightarrow_{\mathrm{rb}}$ can be represented by one special element: its normal form. It follows that $g \leftrightarrow_{\mathrm{rb}} h$ if and only if $N(g) \cong N(h)$.

5. *Axioms*

In this section, we provide a complete axiomatization of branching bisimulation congruence on closed BCCS-expressions (build from the operators 0, +, and Act). A complete axiomatization for (open) BCCS-expressions with recursion (modeling the regular processes) is provided in Van Glabbeek [1993a]. Proof principles to deal with infinite-state processes include the *Recursive Definition Principle* (RDP), saying that every set of recursion equations has a solution; the *Recursive Specification Principle* (RSP), saying that if these equations are

TABLE II. τ-LAWS ($a \in$ ACT).

$a\tau x = ax$	T1
$\tau x = \tau x + x$	T2
$a(\tau x + y) = a(\tau x + y) + ax$	T3

guarded there is only one such solution; the *Approximation Induction Principle* (AIP^-), saying that two processes of which at least one is image-finite are equal iff their finite approximations are equal up to any depth; and *Kooman's Fair Abstraction Rule* ($KFAR^b$), saying that a process will not infinitely often choose an internal action over a visible one. These principles are introduced in Baeten et al. [1987a] in the setting of weak bisimulation semantics. In branching bisimulation semantics, these principles are stated and applied in exactly the same way, except that KFAR has to be slightly reformulated (hence, the superscript b). All principles, including the reformulated KFAR (under the name HAR), can be found in Baeten and Van Glabbeek [1987] in the setting of η-bisimulation semantics. In the context of branching bisimulation semantics, they show up in Baeten and Weijland [1990].

Milner [1980] found that the algebra $(G^p/\underline{\leftrightarrow}_{rw}, 0, +, \mathrm{Act})$ can be completely axiomatized by BCCS (see Section 3) together with the three equations shown in Table II.

THEOREM 5.1. *BCCS + T1 – T3 is a complete axiomatization of* $(G^p/\underline{\leftrightarrow}_{rw}, 0, +, Act)$.

From Figure 1, one can observe that the constructions (b) and (c) are highly fundamental for the behavior of τ in the graph model. For instance, by simplifying Figure 1(b), one finds the second τ-law T2, whereas T3 can be easily found from Figure 1(c). This shows us that the extra τ-laws T2 and T3 originate from the fact that observation equivalence does not preserve branching structures.

Since branching bisimulation equivalence distinguishes between all three graphs in Figure 1, we expect that the laws T2 and T3 will no longer hold in $(G^p/\underline{\leftrightarrow}_{rb}, 0, +, \mathrm{Act})$. As it turns out, axiom T3 is complete dropped and T2 (with T1) is considerably weakened to axiom B (see Table III). *B* implies T1 and is derivable from T1 and T2, as one can check easily. This axiom originates from the axiomatization of η, a constant for abstraction from Baeten and Van Glabbeek [1987] similar to τ. There it appeared as the second η-law H2. In this section, we will establish that:

THEOREM 5.2. *BCCS + B is a complete axiomatization of* $(G^p/\underline{\leftrightarrow}_{rb}, 0, +, Act)$.

Obviously, as $\underline{\leftrightarrow}_{r\eta}$ is a coarser notion than $\underline{\leftrightarrow}_{rb}$ it respects the axiom B. Baeten and Van Glabbeek [1987] established that rooted η-bisimulation equivalence is completely axiomatized by B and T3.

THEOREM 5.3. *BCCS + B + T3 is a complete axiomatization of* $(G^p/\underline{\leftrightarrow}_{r\eta}, 0, +, Act)$.

Finally, a completeness theorem for delay bisimulation was established by Walker [1990].

TABLE III. τ-LAW FOR BRANCHING BISIMULATION.

$a(\tau(y+z)+y)=a(y+z)$	B

$$\begin{tikzcd}
 & \substack{\text{weak bisimulation}\\ \text{T1, T2, T3}} \arrow[dl] \arrow[dr] & \\
\substack{r\eta\text{-bisimulation}\\ \text{B, T3}} \arrow[dr] & & \substack{\text{delay bisimulation}\\ \text{T1, T2}} \arrow[dl] \\
 & \substack{\text{branching bisimulation}\\ \text{B}} &
\end{tikzcd}$$

FIG. 4. Four notions of bisimulation with τ.

THEOREM 5.4. *BCCS + T1 − T2 is a complete axiomatization of* $(G^p/\underline{\leftrightarrow}_{\mathrm{rd}}, 0, +, Act)$.

Hence, we have the diagram shown in Figure 4.

We now present the proofs of the Completeness Theorems 5.1–5.4, employing the method of *graph transformations*, due to Bergstra and Klop [1985]. Most of the section is devoted to our own result, 5.2, but the last part shows how this result can be used to give shorter proofs of the existing Theorems 5.1, 5.3, and 5.4. The basic idea in the proofs is to establish a *graph rewriting system* on finite process graphs, which is confluent and terminating. We prove that (i) normal forms with respect to the graph rewriting system are normal forms in the sense of Definition 4.12; hence, two normal forms are bisimilar iff they are equal (i.e., isomorphic). Furthermore, we prove that every rewriting step in the system (ii) preserves bisimulation, and (iii) corresponds to a proof step in the theory. Then we conclude:

— two finite graphs are bisimilar iff they have the same normal form;
— if two graphs have the same normal form then the corresponding terms can be proved equal. To start with, let us consider some definitions.

Definition 5.5. Let $H \subseteq G$ be the set of *finite* process graphs, having only finitely many paths.

Note that finite process graphs are acyclic and thus certainly root-unwound, and contain only finitely many nodes and edges. Later on, we will establish a correspondence between graphs from H and closed terms in BCCS. Below, we will use the results from the previous section, starting from Proposition 4.11.

Definition 5.6. A pair (r, s) of nodes in a process graph g is called a pair of *double nodes* if $r \neq s$, $r, s \neq \mathrm{root}(g)$ and for all nodes t and labels $a \in \mathrm{Act}$: $r \to^a t \Leftrightarrow s \to^a t$ (see Figure 5).

Definition 5.7. An edge $r \to^\tau s$ in a process graph g is called *manifestly inert* if $r \neq \mathrm{root}(g)$ and for all nodes t and labels $a \in \mathrm{Act}$ such that $(a, t) \neq (\tau, s)$: $r \to^a t \Rightarrow s \to^a t$.

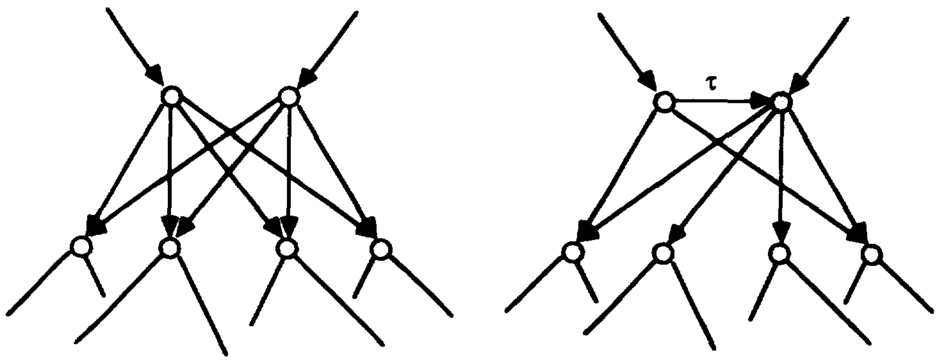

FIG. 5. A pair of double nodes (left) and a manifestly inert τ-step.

Note that for finite process graphs g, the requirement $r, s \neq \text{root}(g)$ in Definition 5.6 is redundant. A τ-edge in a root unwound graph g is *inert* if it is between two rooted branching bisimilar nodes (i.e., nodes that have the same color in the canonical coloring of g). For root unwound graphs, it is easily checked that if (r, s) is a pair of double nodes or if $r \rightarrow^{\tau} s$ is manifestly inert, then r and s are rooted branching bisimilar (though it need not be the case that $(g)_r$ and $(g)_s$ are rooted branching bisimilar). As one can see from Figure 5, it is essential in the Definitions 5.6 and 5.7 that this can be found by investigating the outgoing edges only up to one level. For this reason, in Definition 5.7, the τ-step is called *manifestly* inert, since it can be recognized as such. On $\boldsymbol{H}$, sharing of double nodes and contraction of manifestly inert τ-steps turns out to be strong enough to reduce a graph to its normal form. This means that in the reduction process all rooted branching bisimilar nodes become *manifestly* rooted branching bisimilar.

THEOREM 5.8. *A graph $g \in \boldsymbol{H}$ without double nodes or manifestly inert edges is in normal form.*

PROOF. Let $g \in \boldsymbol{H}$ be a finite graph that is not in normal form. Then, with respect to its canonical coloring (Proposition 4.11), it has at least one pair of different nodes with the same color. Now define the *depth* $d(s)$ of a node s as the number of edges in the longest path starting from s, and the *combined depth* of two nodes as the sum of their depths. Choose a pair (r, s) of different equally colored nodes in g with minimal combined depth. Consequently, we have:

if r' and s' have the same color and $d(r') + d(s') < d(r) + d(s)$, then $r' = s'$. (*)

Without loss of generality assume $d(s) \leq d(r)$. Then, we prove the following two statements:

(1) *if* $r \rightarrow^{a} t$ $(a \in \text{Act})$ *and* $(a, t) \neq (\tau, s)$, *then* $s \rightarrow^{a} t$;
(2) *if* $s \rightarrow^{a} t$ $(a \in \text{Act})$, *then either* $r \rightarrow^{\tau} s$ *or* $r \rightarrow^{a} t$.

From these two statements, we find that, if $r \to^{\tau} s$ is an edge in g, then it is manifestly inert, and if $r \to^{\tau} s$ is not an edge in g, then (r, s) is a pair of double nodes, which proves our theorem. Note that since r and s are different equally colored nodes, they both must be different from the root.

ad 1. Let $r \to^{a} t$ be an edge in g and $(a, t) \neq (\tau, s)$. Since r and s have the same color (hence, the same colored traces), we find that either $a = \tau$ and t has the same color as r and s, or s has the colored trace $(C(r), a, C(t))$. In the first case, it follows from $d(t) < d(r)$ and $(*)$ that $t = s$, which is in contradiction with our assumption $(a, t) \neq (\tau, s)$. So s has a colored trace $(C(r), a, C(t))$. Suppose that $s \to^{\tau} u$ for a node u with color $C(u) = C(s) = C(r)$, then it follows from $d(u) < d(s)$ and $(*)$ that $u = r$, contradicting $d(u) < d(s) \leq d(r)$. Hence, there is a node u such that $s \to^{a} u$ and $C(t) = C(u)$, and since $d(t) + d(u) < d(r) + d(s)$ we conclude from $(*)$ that $t = u$. Hence, $s \to^{a} t$ is an edge in g.

ad 2. Let $s \to^{a} t$ be an edge in g. If $C(t) = C(s) = C(r)$, then it follows from $(*)$ and $d(t) < d(s)$ that $r = t$, in contradiction with $d(t) < d(s) \leq d(r)$. So $(C(s), a, C(t))$ is a colored trace of s, and since r and s have the same color $(C(s), a, C(t))$ is a colored trace of r as well. Now if r has an outgoing τ-edge $r \to^{\tau} u$ to a node with the same color $C(r)$, then it follows from $d(u) + d(s) < d(r) + d(s)$ and $(*)$ that $u = s$. If r has no such edge, then it has an edge $r \to^{a} u$ with $C(u) = C(t)$, and since $d(u) + d(t) < d(r) + d(s)$ we find that $u = t$. Thus, we proved that either $r \to^{\tau} s$ or $r \to^{a} t$, which proves (2). □

Theorem 5.8 tells us that all we need do in order to turn a finite graph g into its normal form is to repeatedly unify its pairs of double nodes and contract its manifestly inert edges. In the case of finite graphs, this can be done in finitely many steps as follows:

Definition 5.9. For any graph $g \in \boldsymbol{H}$ the rewriting relation $\to_{H}$ is defined by the following two one-step reductions:

(1) *sharing* a pair of double nodes (r, s): replace all edges $t \to^{a} r$ by $t \to {}^{a}s$ (if not already there; otherwise, just remove $t \to^{a} r$) and remove r together with all its outgoing edges from g;
(2) *contracting* a manifestly inert step $r \to^{\tau} s$: replace all edges $t \to^{a} r$ by $t \to^{a} s$ (if not already there; otherwise just remove $t \to^{a} r$) and remove r together with all its outgoing edges from g.

PROPOSITION 5.10. *The rewriting relation* $\to_{H}$ *has the following properties*:

(*i*) ***H*** *is closed under applications of* $\to_{H}$
(*ii*) *if* $g \to_{H} h$ *then* $g \ \underline{\leftrightarrow}_{rb}\ h$
(*iii*) $\to_{H}$ *is confluent and terminating.*

PROOF

(i) In applications of $\to_{H}$ the root is never removed and in the resulting graph all nodes remain accessible from the root. It is never the case that two edges with the same label appear between the same two nodes. The graph also remains finite.
(ii) Suppose (r, s) is a pair of double nodes or $r \to^{\tau} s$ is a manifestly inert edge in g, and $g \to_{H} h$ identifies the nodes r and s (= removes the node

$r\rangle$. Let I be the identity relation on the nodes of h, the $I \cup \{(r, s)\}$ is a rooted branching bisimulation between g and h. This is easy to prove from the Definitions 5.6 and 5.7.

(iii) $\rightarrow_H$ is terminating since it decreases the number of nodes, and every finite process graph has finitely many nodes. Next, suppose g has two normal forms n and n', then by the definition of $\rightarrow_H$ n and n' are without pairs of double nodes and without manifestly inert edges. Thus by Theorem 5.8 n and n' are in normal form. By (ii), it follows that $n \leftrightarrow_{rb} n'$ and hence by Theorem 4.16 (normal form theorem) we have $n \cong n'$. □

Next we will establish a correspondence between finite graphs and closed BCCS-terms, such that the graph reductions of Definition 5.9 correspond to proof steps in BCCS + B.

Write $s \equiv_\Gamma t$ for $\Gamma \vdash s = t$, saying that s and t are equal *modulo* applications of axioms from Γ and the standard axioms for equality (reflexivity, commutativity, transitivity, and replacement). Let T(BCCS) be the set of closed BCCS terms. It is quite easy to turn finite graphs into such terms:

Definition 5.11. Let $\langle \cdot \rangle$: $\boldsymbol{H} \rightarrow T(\text{BCCS})$ be a mapping that satisfies

$$\langle g \rangle = \sum_{r(g) \rightarrow^a s \text{ is an edge in } g} a.\langle (g)_s \rangle.$$

Here $r(g)$ denotes the root node of $g \in \boldsymbol{H}$ and if p_i is a BCCS-term for $i \in I$, with $I = \{i_1, \ldots, i_n\}$ a finite nonempty set of indices, then $\Sigma_{i \in I}\, p_i$ denotes a BCCS-term $p_{i_1} + \cdots + p_{i_n}$. In case $I = \varnothing$ the term 0 is denoted. Note that the notation $\Sigma_{i \in I}\, p_i$ does not determine the order and association of the terms p_i.

If $g \in \boldsymbol{H}$ and $r(g) \rightarrow^a s$ is an edge in g, then $(g)_s \in \boldsymbol{H}$. Furthermore, since $g \in \boldsymbol{H}$ is finite, $r(g)$ has only finitely many outgoing edges, so the requirement of Definition 5.11 is well-defined. Moreover, with induction on the depth of its argument, one easily proves that a mapping that meets this requirement exists. However, for $g \in \boldsymbol{H}$, this requirement determines $\langle g \rangle$ only modulo A1–A2.

PROPOSITION 5.12. *if* $g, h \in \boldsymbol{H}$ *and* $g \cong h$, *then* A1–A2 $\vdash \langle g \rangle = \langle h \rangle$.

PROOF. Trivial. □

Definition 5.13. The denotation $[\![p]\!]$ of a BCCS-term p in the graph domain $\boldsymbol{G}$, is defined by:

$$[\![0]\!] = 0_G$$
$$[\![a.x]\!] = a_G([\![x]\!]) \qquad \text{for} \quad a \in \text{Act}$$
$$[\![x + y]\!] = [\![x]\!] +_G [\![y]\!]$$

where 0_G, a_G and $+_G$ are the interpretations in $\boldsymbol{G}$ of the constants and operators 0, a and + of BCCS, as defined in Definition 3.1.

The following proposition says that $\langle \cdot \rangle$ is a left-inverse of $[\![\cdot]\!]$, modulo A0, A1, and A2.

PROPOSITION 5.14. *If $p \in T(BCCS)$, then $\langle [\![p]\!] \rangle \equiv_{A0\text{-}2} p$.*

PROOF. With induction to the structure of terms:

— If $p = 0$, then $[\![p]\!]$ is the trivial graph, and $\langle [\![p]\!] \rangle = p$.
— If $p = a.u$ for some BCCS-term u, then $[\![p]\!]$ is the graph with an edge labeled a followed by $[\![u]\!]$. Then $\langle [\![p]\!] \rangle = a.\langle [\![u]\!] \rangle$ and so by induction we find that $\langle [\![p]\!] \rangle \equiv_{A0\text{-}2}$ a.u.
— Suppose $p = u + v$. One can easily see that for graphs g and h: $\langle g +_G h \rangle \equiv_{A0\text{-}2} \langle g \rangle + \langle h \rangle$. Thus: A0-A2 $\vdash \langle [\![u + v]\!] \rangle = \langle [\![u]\!] \rangle + \langle [\![v]\!] \rangle = u + v$ (by induction). □

This concludes the establishment of a correspondence between $\boldsymbol{H}$ and T(BCCS). Next, we will show that every rewriting step on $\boldsymbol{H}$ corresponds to a proof step in BCCS + B.

LEMMA 5.15. *Let (r, s) be a pair of double nodes or $r \to^{\tau} s$ be a manifestly inert τ-step in a process graph g and let $a \in Act$. Then we have: BCCS + B $\vdash a.\langle (g)_r \rangle = a.\langle (g)_s \rangle$.*

PROOF. In case (r, s) is a pair of double nodes, r has an edge $r \to^{a} t$ iff s has an edge $s \to^{a} t$, and so $\langle (g)_r \rangle \equiv_{A1,2} \langle (g)_s \rangle$; hence, $a.\langle (g)_r \rangle = a.\langle (g)_s \rangle$.

In case $r \to^{\tau} s$ is a manifestly inert τ-step there must be BCCS terms p and q such that

(1) $\langle (g)_r \rangle \equiv_{A0\text{-}2} \tau.\langle (g)_s \rangle + p$
(2) $\langle (g)_s \rangle \equiv_{A0\text{-}2} p + q$.

So we derive:

$$\begin{aligned} \text{A0-2 + B} \vdash a \cdot \langle (g)_r \rangle &= a.(\tau.\langle (g)_s \rangle + p) && \text{(by (1))} \\ &= a.(\tau.(p + q) + p) && \text{(by (2))} \\ &= a.(p + q) && \text{(by applying B)} \\ &= a.\langle (g)_s \rangle && \text{(by (2)).} \end{aligned}$$ □

PROPOSITION 5.16. *If $g \to_H h$, then BCCS + B $\vdash \langle g \rangle = \langle h \rangle$.*

PROOF. On $\boldsymbol{H}$ the rewriting relation $\to_H$ can be decomposed in the following elementary reductions: "Take a pair of double nodes (r, s) or a manifestly inert τ-step $r \to^{\tau} s$ and replace one edge $t \to^{a} r$ by $t \to^{a} s$ (if not already there; otherwise, just remove $t \to^{a} r$) and if r has no more incoming edges remove r together with all its outgoing edges from g". So it suffices to prove that if h is obtained from g by means of such an elementary reduction, we have $\langle g \rangle \equiv_{\Gamma} \langle h \rangle$, where $\Gamma = BCCS$ + B. From Definition 5.11, it follows that it even suffices to prove $\langle (g)_t \rangle \equiv_{\Gamma} \langle (h)_t \rangle$.

— First consider the case that there is no edge $t \to^{a} s$ in g. Then $\langle (g)_t \rangle \equiv_{A0\text{-}2} a.\langle (g)_r \rangle + p$ for certain term p. Lemma 5.15 says $a.\langle (g)_r \rangle \equiv_{\Gamma} .\langle (g)_s \rangle$; hence,

$$\langle (g)_t \rangle \equiv_{\Gamma} a.\langle (g)_s \rangle + p \equiv_{A0\text{-}2} \langle (h)_t \rangle.$$

— In case $t \to^a s$ is an edge in g we have

$$\langle (g)_t \rangle \equiv_{\mathrm{A0\text{-}2}} a.\langle (g)_r \rangle + a.\langle (g)_s \rangle + p \equiv_{\Gamma}$$
$$a.\langle (g)_s \rangle + a.\langle (g)_s \rangle + p \equiv_{\mathrm{A2,3}} a.\langle (g)_s \rangle + p \equiv_{\mathrm{A0\text{-}2}} \langle (h)_t \rangle. \qquad \square$$

Now we are in the position to prove the completeness of BCCS + B with respect to $\boldsymbol{G}^p/\underline{\leftrightarrow}_{\mathrm{rb}}$:

Proof of Theorem 5.2

(soundness). The fact that $(\boldsymbol{G}^p/\underline{\leftrightarrow}_{\mathrm{rb}}, 0, +, \mathrm{Act})$ is a model for BCCS + B follows easily by inspection of the axioms.

(completeness). Let $(\boldsymbol{G}^p/\underline{\leftrightarrow}_{\mathrm{rb}}, 0, +, \mathrm{Act}) \models p = q$ for two closed BCCS-terms p and q, then by definition $[\![p]\!] \underline{\leftrightarrow}_{\mathrm{rb}} [\![q]\!]$. Let g and h be the unique normal forms of $[\![p]\!]$ and $[\![q]\!]$ with respect to $\to_H$. By Proposition 5.10, we find $g \underline{\leftrightarrow}_{\mathrm{rb}} [\![p]\!] \underline{\leftrightarrow}_{\mathrm{rb}} [\![q]\!] \underline{\leftrightarrow}_{\mathrm{rb}} h$. From Theorem 5.8 it follows that g and h must be in normal form in the sense of Section 4 and by the normal form theorem (4.16) it then follows that $g \cong h$. Thus, we find BCCS + $B \vdash p = \langle [\![p]\!] \rangle = \langle g \rangle = \langle h \rangle = \langle [\![q]\!] \rangle = q$ using Propositions 5.12, 5.14, and 5.16. So BCCS + B is a complete axiomatization of $\boldsymbol{G}^p/\underline{\leftrightarrow}_{\mathrm{rb}}$. $\square$

Next, we will prove the other completeness theorems, using the earlier results in this section. In fact, we will extend the graph rewriting system to one which is "typical" for the corresponding bisimulation relation. The rewrite rules which are added to the system are derived from Figure 1: In case of η-bisimulation, we will *saturate* the graph by exhaustively adding edges of the kind of Figure 1(c), whereas in the case of delay bisimulation we add edges as in Figure 1(b). For weak bisimulation, we do both. This way, we obtain normal forms which are saturated and which turn out to be unique modulo rooted branching bisimulation. From there, we establish the completeness result precisely in the same way as before.

Definition 5.17. Let $a \in \mathrm{Act}$, then:

(1) The rewriting relation $\to_\eta$ is defined on $\boldsymbol{H}$ by the rule:
if a graph has a path $s \to^a s_1 \to^\tau s'$ without an edge $s \to^a s'$, then add $s \to^a s'$.

(2) The rewriting relation $\to_d$ is defined on $\boldsymbol{H}$ by the rule:
if a graph has a path $s \to^\tau s_1 \to^a s'$ without an edge $s \to^a s'$, then add $s \to^a s'$.

(3) Furthermore, we set: $\to_\tau = \to_\eta \cup \to_d$.

Applications of $\to_\eta$, $\to_d$, or $\to_\tau$ are referred to as *saturation steps* (cf. Bergstra and Klop [1988]).

Proposition 5.18. *The relations* $\to_\eta$, $\to_d$, *and* $\to_\tau$ *satisfy the following properties*:

(*i*) $\boldsymbol{H}$ *is closed under applications of* $\to_\eta$, $\to_d$, *and* $\to_\tau$

(*ii*) $\to_\eta$, $\to_d$, *and* $\to_\tau$ *are confluent and terminating.*

Proof

(i) Directly from Definition 5.17.

(ii) (termination). Let $g \in \boldsymbol{H}$. Let $n(g)$ be the (finite) number of nodes in g, $l(g)$ be the number of labels and $e(g)$ be the number of edges in g. Note

that $n(g)$ and $l(g)$ are not changed by $\rightarrow_\eta$, $\rightarrow_d$, and $\rightarrow_\tau$ whereas $e(g)$ increases with every saturation step. Since g is finite we find that $e(g) < n(g) \times l(g) \times n(g)$ and so $n(g) \times l(g) \times n(g) - e(g)$ is positive and decreasing with the number of saturation steps.
(confluence). $\rightarrow_\eta$, $\rightarrow_d$, and $\rightarrow_\tau$ do not eliminate redexes. □

So, from Proposition 5.18, we find that any graph $g \in \boldsymbol{H}$ has unique normal forms with respect to $\rightarrow_\eta$, $\rightarrow_d$, and $\rightarrow_\tau$. These are written as $H(g)$, $D(g)$, and $T(g)$ and (in that order) are called η-, d-, and *τ-saturated*. The latter is also often referred to as the *transitive closure* of τ-steps. Furthermore, as is straightforward to prove, saturation preserves the corresponding bisimulation:

PROPOSITION 5.19. *For all $g, h \in \boldsymbol{H}$:*

(*i*) *if $g \rightarrow_\eta h$, then $g \underline{\leftrightarrow}_{r\eta} h$;*
(*ii*) *if $g \rightarrow_d h$, then $g \underline{\leftrightarrow}_{rd} h$;*
(*iii*) *if $g \rightarrow_\tau h$, then $g \underline{\leftrightarrow}_{rw} h$.*

THEOREM 5.20 (NORMAL FORM THEOREM). *Let $g, h \in \boldsymbol{H}$, then*

(*i*) *if g and h are η-saturated process graphs, then $g \underline{\leftrightarrow}_{r\eta} h$ if and only if $g \underline{\leftrightarrow}_{rb} h$;*
(*ii*) *if g and h are d-saturated process graphs, then $g \underline{\leftrightarrow}_{rd} h$ if and only if $g \underline{\leftrightarrow}_{rb} h$;*
(*iii*) *if g and h are τ-saturated process graphs, then $\underline{\leftrightarrow}_{rw} h$ if and only if $g \underline{\leftrightarrow}_{rb} h$.*

PROOF. We will only prove (i). The other cases proceed in the same way.
Suppose that R: $g \underline{\leftrightarrow}_{r\eta} h$, then it is sufficient to prove that R is a rooted branching bisimulation:

(i) The roots of $H(g)$ and $H(h)$ are related and R satisfies the root condition.
If $R(r, s)$ and $r \rightarrow^a r'$ then either $a = \tau$ and $R(r', s)$, or $s \Rightarrow s_1 \rightarrow^a s_2 \Rightarrow s'$ such that $R(r, s_1)$ and $R(r', s')$. Let $t_1, \ldots, t_k$ be such that $s_2 = t_0 \rightarrow^\tau t_1 \rightarrow^\tau \cdots \rightarrow^\tau t_k = s'$ $(k \geq 0)$, then since g and h are η-saturated there are edges $s_1 \rightarrow^a t_i$ and so there is a path $s \Rightarrow s_1 \rightarrow^a s'$. □

COROLLARY 5.21

(*i*) *$g \underline{\leftrightarrow}_{r\eta} h$ if and only if $H(g) \underline{\leftrightarrow}_{rb} H(h)$ if and only if $N(H(g)) \cong N(H(h))$;*
(*ii*) *$g \underline{\leftrightarrow}_{rd} h$ if and only if $D(g) \underline{\leftrightarrow}_{rb} D(h)$ if and only if $N(D(g)) \cong N(D(h))$;*
(*iii*) *$g \underline{\leftrightarrow}_{rw} h$ if and only if $T(g) \underline{\leftrightarrow}_{rb} T(h)$ if and only if $N(T(g)) \cong N(T(h))$.*

PROOF. It follows by Proposition 5.19 that $H(g) \underline{\leftrightarrow}_{r\eta} g$, $D(g) \underline{\leftrightarrow}_{rd} g$ and $T(g) \underline{\leftrightarrow}_{rw} g$. Now apply the normal form Theorems 5.20 and 4.16. □

So we find that in each $r*$-bisimulation equivalence class of finite process graphs for $* \in \{w, \eta, d\}$ there is exactly one $*$-saturated process graph up to rooted branching bisimulation and exactly one $*$-saturated normal form up to isomorphism. In order to prove the completeness theorems, we still need to prove that rewriting steps correspond to proof steps.

PROPOSITION 5.22. *For finite graphs g and h:*

(*i*) *If $g \rightarrow_\eta h$, then $BCCS$ + B,T3 $\vdash \langle g \rangle = \langle h \rangle$;*
(*ii*) *If $g \rightarrow_d h$, then $BCCS$ + T2 $\vdash \langle g \rangle = \langle h \rangle$;*
(*iii*) *If $g \rightarrow_\tau h$, then $BCCS$ + T1–3 $\vdash \langle g \rangle = \langle h \rangle$.*

PROOF

(i) If $r \rightarrow^{a} r' \rightarrow^{\tau} r''$ is a path in g and $r \rightarrow^{a} r''$ is added in g to obtain h, then we find that

$$\langle (g)_r \rangle \equiv_{\mathrm{A1\text{-}3}} \langle (g)_r \rangle + a.\langle (g)_{r'} \rangle \text{ and } \langle (g)_{r'} \rangle \equiv_{\mathrm{A1\text{-}3}} \tau.\langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle$$

and hence:

$$\begin{aligned} \mathrm{BCCS} + \mathrm{T3} \vdash \langle (g)_r \rangle &= \langle (g)_r \rangle + a.(\tau.\langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) \\ &= \langle (g)_r \rangle + a.(\tau.\langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) + a.\langle (g)_{r''} \rangle \quad \text{(by T3)} \\ &= \langle (g)_r \rangle + a.\langle (g)_{r''} \rangle = \langle (h)_r \rangle . \end{aligned}$$

From BCCS + T3 $\vdash \langle (g)_r \rangle = \langle (h)_r \rangle$ it easily follows that BCCS + T3 $\vdash \langle g \rangle = \langle h \rangle$.

(ii) If $r \rightarrow^{\tau} r' \rightarrow^{a} r''$ is a path is g and $r \rightarrow^{a} r''$ is added in g to obtain h, then:

$$\langle (g)_r \rangle \equiv_{\mathrm{A1\text{-}3}} \langle (g)_r \rangle + \tau.\langle (g)_{r'} \rangle \text{ and } \langle (g)_{r'} \rangle \equiv_{\mathrm{A1\text{-}3}} a.\langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle$$

and hence:

$$\begin{aligned} \mathrm{BCCS} + \mathrm{T2} \vdash \langle (g)_r \rangle &= \langle (g)_r \rangle + \tau.(a.\langle (g)_{r''} \rangle + \langle (g)_{r'} \rangle) \\ &= \langle (g)_r \rangle + a.\langle (g)_{r''} \rangle \quad \text{(by T2 and A3)} = \langle (h)_r \rangle . \end{aligned}$$

Hence, BCCS + T2 $\vdash \langle g \rangle = \langle h \rangle$.

(iii) Immediately from (i) and (ii). □

PROOFS OF THE THEOREMS 5.1, 5.3, AND 5.4. The soundness theorems follow easily after inspection of the axioms. Of the completeness theorems, we only prove Theorem 5.3. The others proceed in the same way.

Let $(G^{p}/\underline{\leftrightarrow}_{r\eta}, 0, +, \mathrm{Act}) \models p = q$ for two closed BCCS-terms p, q, then, by definition, $[\![p]\!] \underline{\leftrightarrow}_{r\eta} [\![q]\!]$. Let g and h be the unique normal forms of $[\![p]\!]$ and $[\![q]\!]$ with respect to $\rightarrow_{\eta}$. By proposition 5.19, we find $g \underline{\leftrightarrow}_{r\eta} [\![p]\!] \underline{\leftrightarrow}_{r\eta} [\![q]\!] \underline{\leftrightarrow}_{r\eta} h$. The graphs g and h must be η-saturated and, by the normal form theorem (Theorem 5.20), it then follows that $g \underline{\leftrightarrow}_{rb} h$. Thus, we find BCCS + B,T3 $\vdash p = \langle [\![p]\!] \rangle = \langle g \rangle = \langle h \rangle = \langle [\![q]\!] \rangle = q$ using Propositions 5.14 and 5.22 and Theorem 5.2. So BCCS + B,T3 is a complete axiomatization of $G^{p}/\underline{\leftrightarrow}_{r\eta}$. □

6. *Correspondence*

Here, we present a theorem that tells us that in quite a number of cases observation and branching bisimulation equivalence are the same. For instance, consider the practical applications where implementations are verified by proving them equal to some specification (after having abstracted from a set of unobservable actions of course). In many such cases, the *specification* does not involve any τ-steps at all: in fact, all τ-steps that occur in the verification process originate from the abstraction procedure that is carried out on the implementation.

As it turns out, in all such cases there is no difference between observation and branching bisimulation equivalence. For this reason, we may expect many verifications involving observation equivalence to be valid in the stronger setting of branching bisimulation as well. In particular, this is the case for all protocol verifications in weak bisimulation semantics known to the authors.

THEOREM 6.1. *Suppose g and h are two graphs, and g is without edges labeled with τ. Then*:

(*i*) $g \leftrightarrow_{w} h$ *if and only if* $g \leftrightarrow_{b} h$;
(*ii*) $g \leftrightarrow_{rw} h$ *if and only if* $g \leftrightarrow_{rb} h$.

PROOF. Let r be the *largest* (rooted) weak bisimulation between g and h. We show that R is even a (rooted) branching bisimulation. Assume that $R(r, s)$ and $r \rightarrow^{a} r'$ is an edge in g, then either $a = \tau$ and $R(r', s)$—contradicting the absence of τ-edges in g—or in h there is a path $s \Rightarrow s_1 \rightarrow^{a} s_2 \Rightarrow s'$ and $R(r', s')$. Assume $s \Rightarrow s_1$ has the form $s = v_0 \rightarrow^{\tau} v_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} v_m = s_1$ ($m \geq 0$), then it follows from $s \rightarrow^{\tau} v_1$ and $R(r, s)$ that for some r_1: $r \Rightarrow r_1$ and $R(r_1, v_1)$. Since g has no τ-edges we find that $r = r_1$. Repeating this argument m times, we find that $R(r, v_i)$ and $R(r, s_1)$.

Furthermore, since $R(r, s_1)$ and $s_1 \rightarrow^{a} s_2$ we find that $r \rightarrow^{a} r''$ (g has no τ-steps) such that $R(r'', s_2)$. Since $s_2 = w_0 \rightarrow^{\tau} w_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} w_n = s'$, it follows from the same argument as before that $R(r'', w_i)$ and $R(r'', s')$. Thus, we find $R(r', s')$, $R(s', r'')$ and $R(r'', s_2)$ and since R is the largest rooted weak bisimulation we have $R(r', s_2)$.

On the other hand, if $R(r, s)$ and $r \rightarrow^{a} r'$ is an edge in h, then either $a = \tau$ and $R(r', s)$ or directly $s \rightarrow^{a} s'$ such that $R(r', s')$, since g contains no τ-edges. □

For η-instead of branching bisimulation equivalence this theorem was already proven in Baeten and van Glabbeek [1987]. From Theorem 3.7, we easily find that for graphs g and h:

$$g \text{ is without } \tau\text{-edges} \Rightarrow (\tau.g \leftrightarrow_{rw} \tau.h \Rightarrow \tau.g \leftrightarrow_{rb} \tau.h).$$

Bouali [1992] presents what is claimed to be "a counter example to the feelings in [4] about the fact that Branching Bisimulation and Weak Bisimulation coincide for a large class of processes." Here "[4]" is Groote and Vaandrager [1990], who refer to the results of this section and confirm that they "are not aware of any protocol that can be verified in the setting of observation equivalence but not in the stronger setting of branching bisimulation equivalence." Bouali's counterexample concerns the minimization of Peterson's [1981] mutual exclusion algorithm. Here *minimization* means finding an equivalent process that is as small as possible. In branching bisimulation semantics, this yields a process with 17 states, whereas in weak bisimulation semantics a process with only 14 states is obtained. In our opinion, this does not constitute a counterexample, as this is not a case of verification. It would be a verification if the minimal representation modulo weak bisimulation would be an acceptable *specification*. However, it has too many τ-transitions for this purpose. It is a process obtained as the *result* of a calculation, not anything one likely would start with. Much more plausible candidates for a specification are the minimizations modulo coupled simulation (9 states) or failure equivalence (4 states). More on this matter in the conclusion.

7. *Refinement*

Virtually all semantic equivalences employed in theories of concurrency are—as in this paper—defined in terms of *actions* that concurrent systems may perform. Mostly, these actions are taken to be *atomic*, meaning that they are considered not to be divisible into smaller parts. In this case, the defined equivalences are said to be based on *action atomicity.*

However, in the top-down design of distributed systems it might be fruitful to model processes at different levels of abstraction. The actions on an abstract level then turn out to represent complex processes on a more concrete level. This methodology does not seem compatible with nondivisibility of actions and for this reason Pratt [1986], Lamport [1986], and others plead for the use of semantic equivalences that are not based on action atomicity.

As indicated in Castellano et al. [1987], the concept of action atomicity can be formalized by means of the notion of *refinement of actions.* A semantic equivalence is *preserved under action refinement* if two equivalent processes remain equivalent after replacing all occurrences of an action a by a more complex process $r(a)$. In particular, $r(a)$ may be a sequence of two actions a_1 and a_2. An equivalence is strictly based on action atomicity if it is not preserved under action refinement.

In the previous sections of this paper, we argued that Milner's notion of observation equivalence does not respect the branching structure of processes, and proposed the finer notion of branching bisimulation equivalence which does. In this section, we moreover find, that observation equivalence is not preserved under action refinement, whereas branching equivalence is.

From the axiom T3 (see Table II), it is easy to show why the notion of observation congruence is not preserved under refinement of actions: replacing the action a by the process bc, we obtain $bc(\tau x + y) = bc(\tau x + y) + bcx$, which obviously is not valid in $G^p/\underline{\leftrightarrow}_{rw}$.

In this section, we will prove that branching equivalence *is* preserved under refinement of actions, and so it allows us to look at actions as abstractions of much larger structures. As is common in the area of action refinement, we exclude refining an action into an empty process. Allowing this kind of refinement would not be compatible with any of the abstract bisimulations considered here, as we have $a\tau 0 + b0 = a0 + b0$ but $\tau 0 + b0 \neq 0 + b0 = b0$. Put $A = \text{Act} \setminus \{\tau\}$. Consider the following definitions.

Definition 7.1 (*substitution*). Let r: $A \to G^p \setminus \{0\}$ be a mapping from observable actions to nontrivial, root-unwound graphs, and suppose $g \in G^p$. Then, the graph $r(g)$ can be found as follows: For every edge $s \to^a s'$ $(a \in A)$ in g, take a copy $\underline{r(a)}$ of $r(a)$ $(\in G^p)$. Next, identify s with the root note of $\underline{r(a)}$, and s' with all endnodes of $\underline{r(a)}$, and remove the edge $s \to^a s'$.

Note that in this definition it is never needed to identify s and s', since $r(a)$ is nontrivial (cf. Definition 3.1). This way, the mapping r is extended to the domain G^p. Note that since $\tau \notin A$, τ-edges cannot be substituted by graphs. Finally, observe that every node in g is a node in $r(g)$.

Definition 7.2 (*preservation under action refinement*). An equivalence $\approx$ on G^p is said to be *preserved under refinement of actions* if for every mapping r: $A \to G^p \setminus \{0\}$, we have: $g \approx h \Rightarrow r(g) \approx r(h)$.

In other words, an equivalence $\approx$ is preserved under refinement if it is a congruence with respect to every substitution operator r.

Starting from a relation R: $g \underline{\leftrightarrow}_{rb} h$, we construct a branching bisimulation $r(R)$: $r(g) \underline{\leftrightarrow}_{rb} r(h)$, proving that preserving branching congruence, every edge with a label from A can be replaced by a root unwound nontrivial graph.

Definition 7.3. Let r: $A \to G^p \setminus \{0\}$ be a mapping from observable actions to graphs, $g, h \in G^p$ and R: $g \leftrightarrow_{\text{rb}} h$. Now $r(R)$ is the smallest relation between nodes of $r(g)$ and $r(h)$, such that:

(1) $R \subseteq r(R)$;

(2) If $r \to^a r'$ and $s \to^a s'$ $(a \in A)$ are edges in g and h such that $R(r, s)$ and $R(r', s')$, and both edges are replaced by copies $\underline{r(a)}$ and $\overline{r(a)}$ of $r(a)$, respectively, then nodes from $\underline{r(a)}$ and $\overline{r(a)}$ are related by $r(R)$ iff they are copies of the same node in $r(a)$.

Edges $r \to^a r'$ and $s \to^a s'$ $(a \in A)$ such that $R(r, s)$ and $R(r', s')$, will be called *related* by R, as well as the copies $\underline{r(a)}$ and $\overline{r(a)}$ that are substituted for them. Observe that on nodes from g and h the relation $r(R)$ is equal to R. Note that if $r(R)(r, s)$, then r is a node in g iff s is a node in h.

THEOREM 7.4 (REFINEMENT). *Branching congruence is preserved under refinement of actions.*

PROOF. We prove that R: $g \leftrightarrow_{\text{rb}} h \Rightarrow r(R)$: $r(g) \leftrightarrow_{\text{rb}} r(h)$ by checking the requirements. For convenience, in the definition of branching equivalence (Definition 2.4), we omit the requirement of the existence of a path $s_2 \Rightarrow s'$, as it is redundant (see the remark just after Definition 2.8). Then, we find:

(i) The root nodes of $r(g)$ and $r(h)$ are related by $r(R)$.

(ii) Assume $r(R)(r, s)$ and in $r(g)$ there is an edge $r \to^a r'$. Then there are two possibilities (similarly in case $r \to^a r'$ stems from $r(h)$):

(1) The nodes r and s originate from g and h. Then $R(r, s)$, and by the construction of $r(g)$, we find that either $a = \tau$ and $r \to^\tau r'$ was already an edge in g, or g has an edge $r \to^b r^*$ and $r \to^a r'$ is a copy of an initial edge from $r(b)$.

In the first case, it follows from R: $g \leftrightarrow_{\text{rb}} h$ that either $R(r', s)$—hence, $r(R)(r', s)$—or in h there is a path $s \Rightarrow s_1 \to^\tau s'$ such that $R(r, s_1)$ and $R(r', s')$. By definition of refinement, the same path also exists in $r(h)$, and thus we have $r(R)(r, s_1)$ and $r(R)(r', s')$.

In the second case, there must be a corresponding path $s \Rightarrow s_1 \to^b s^*$ in h such that $R(r, s_1)$ and $R(r^*, s^*)$. Then, in $r(h)$, we find a path $s \Rightarrow s_1 \to^a s'$ (by replacing $\to^b$ by $r(b)$) such that $r(R)(r, s_1)$ and $r(R)(r', s')$.

(2) The nodes r and s originate from related copies $\underline{r(b)}$ and $\overline{r(b)}$ of a substituted graph $r(b)$ (for some $b \in A$), and are no copies of root or endnodes in $r(b)$. Then $r \to^a r'$ is an edge in $\underline{r(b)}$. From $r(R)(r, s)$ we find that r and s are copies of the same node from $r(b)$. So, there is an edge $s \to^a s'$ in $\overline{r(b)}$ where s' is a copy of the node in $r(b)$, corresponding with r'. Clearly, $r(R)(r', s')$.

(iii) Since for nodes from g and h, we have $r(R)(r, s)$ iff $R(r, s)$, the root condition is satisfied. □

In the framework of ACP (cf. Bergstra and Klop [1984]), where a sequential composition operator is present, the refinement theorem for closed terms (or finite process graphs) can be proved much easier by syntactic analysis of proofs, instead of working with equivalences between graphs. The ACP-version of the axioms for branching bisimulation (cf. Van Glabbeek and Weijland [1991]) does *not* contain any occurrences of (atomic) actions from A. Now assume we have a proof of some equality $s = t$ between closed terms, then this proof consists of a sequence of applications of these axioms. Since all these axioms are universal equations without actions from A, the actions from s and t can be replaced by general variables, and the proof will still hold. Hence, every equation is an instance of a universal equation *without* any actions. Immediately, we find that we can substitute arbitrary closed terms for these variables, obtaining refinement for closed terms.

Nevertheless, the semantic proof of the refinement theorem is important, since it also holds for larger graphs that cannot be represented by closed terms.

Delay bisimulation is also preserved under action refinement (as pointed out in Devillers [1992]), and η-bisimulation is not (same counterexample as for weak bisimulation). Moreover, delay bisimulation is the coarsest equivalence that is preserved under refinement and finer then τ-bisimulation (i.e., *fully abstract* with respect to weak bisimulation and action refinement) [Cherief and Schnoebelen 1991], but branching bisimulation is not quite the coarsest equivalence that is preserved under refinement and finer then η-bisimulation. This was established in Cherief [1992], who characterized the *quasi-branching bisimulation equivalence* below as the coarsest equivalence that is preserved under refinement and finer then η-bisimulation.

Definition 7.5. Two graphs g and h are *quasi-branching bisimilar*—notation: $g \underline{\leftrightarrow}_{qb} h$—if there exists a symmetric relation R (called a *quasi-branching bisimulation*) between the nodes of g and h such that:

(i) The roots are related by R;
(ii) If $R(r,s)$ and $r \to^{a} r'$, then either $a = \tau$ and there is a path $s \Rightarrow s'$ such that $R(r',s')$, or there exists a path $s \Rightarrow s_1 \to^{a} s'$ such that $R(r,s_1)$ and $R(r',s')$.

It is easy to see that quasi-branching bisimulation is finer than both delay and η-bisimulation, but coarser than branching bisimulation equivalence, that is, we have $g \underline{\leftrightarrow}_{b} h \Rightarrow g \underline{\leftrightarrow}_{qb} h \Rightarrow g \underline{\leftrightarrow}_{\eta} h$ and $g \underline{\leftrightarrow}_{qb} h \Rightarrow g \underline{\leftrightarrow}_{d} h$. The strictness of the first implication follows from the example $\tau(\tau + b) + a \underline{\leftrightarrow}_{qb} \tau(\tau + b) + a + \tau$, which does not apply to $\underline{\leftrightarrow}_{b}$. The strictness of the other two implications follows from the fact that neither $\underline{\leftrightarrow}_{\eta}$ nor $\underline{\leftrightarrow}_{d}$ implies the other. Cherief's quasi-branching bisimulation coincides with the *simple branching bisimulation* independently proposed by van Benthem et al. [1993] and discussed in the next section.

Finally, it should be noted that action refinement as defined in this section is a meaningful notion that can be used in the design of systems *only* if these systems are assumed to be sequential (i.e., performing only one action at a time). In the presence of parallel composition, process graphs as presented here are not sufficiently expressive for defining a refinement operator. For this purpose, one may better use causality based models of concurrency, such as event structures or Petri nets (cf. van Glabbeek and Goltz [1990]). Our results imply that no equivalence on such a model that is a generalization of weak bisimulation in the sense that it coincides with weak congruence for sequential processes, can be preserved under action refinement, whereas generalizations of branching bisimulation can be preserved.

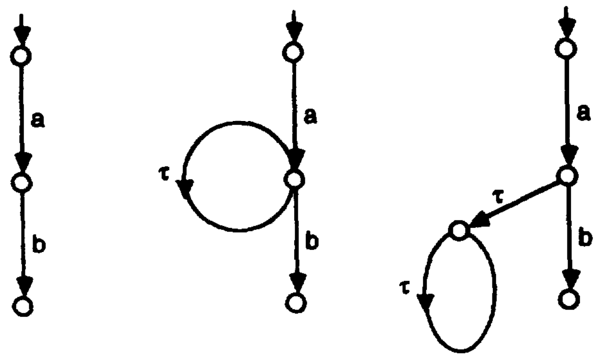

FIG. 6. Three ways of modeling divergence.

8. *Divergence*

In the literature on bisimulation semantics, roughly three ways are suggested for treating divergence (= infinite τ-paths). The original notion of weak bisimulation equivalence [Hennessy and Milner 1980; Milner 1980; Park 1981] abstracted from all divergencies; the first two graphs of Figure 6 are equivalent.

These identifications can be justified by an appeal to fairness [Milner 1980; Baeten et al. 1987a], and play a crucial role in many protocol verifications. In Bergstra et al. [1987], the corresponding semantics is referred to as *bisimulation semantics with fair abstraction.* A variant were divergence is taken into account, in the sense that the first two graphs of Figure 6 are distinguished, was proposed in Hennessy and Plotkin [1980] for (a variant of) weak bisimulation and in Milner [1981] for delay bisimulation. In both cases, a complete axiomatization is provided in Walker [1990]. In these semantics, the basic notion is a *preorder* rather then an equivalence, and divergence is identified with underspecification. The induced equivalences identify the last two graphs of Figure 6, which are distinguished in weak bisimulation semantics with fair abstaction. Hence, the two notions are incomparable. A semantics that refines both notions was proposed in Bergstra et al. [1987] under the name *bisimulation semantics with explicit divergence.*

η-, delay, and branching bisimulation as presented in this paper are all based on the variant of weak bisimulation with fair abstraction. However, it is completely straightforward to generalize the *weak bisimulation preorder* of Hennessy and Plotkin [1980] to a *η-bisimulation preorder*, and the *delay bisimulation preorder* of Milner [1981] to a *branching bisimulation preorder.* Also it is not difficult to define *η-, delay,* and *branching bisimulation with explicit divergence* in the spirit of Bergstra [1987]. For branching bisimulation, the definition can conveniently be given in terms of colored traces.

Definition 8.1. A node in a colored graph is *divergent* if it is the starting point of an infinite path of which all nodes have the same color. A coloring *preserves divergence* if no divergent node has the same color as a nondivergent node. Two graphs g and h are *(rooted) branching bisimulation equivalent with explicit divergence* if there exists a (rooted) consistent divergence preserving coloring on g and h for which they have the same colored trace set.

9. *Modal Characterizations*

It is well known (cf. Hennessy and Milner [1985]) that strong and weak equivalence can be characterized by means of a simple modal language, called Hennessy–Milner logic (HML):

Definition 9.1. The formulas of the *Hennessy–Milner logic* (*HML*) are given by the syntax

$$\varphi ::= \langle a \rangle \varphi \mid \varphi \wedge \psi \mid \neg \varphi \mid T$$

The (strong) satisfaction relation $\models$ between processes and formulas is defined by

(i) $g \models T$	for all process graphs g,
(ii) $g \models \langle a \rangle \varphi$	if there is an edge $\mathrm{root}(g) \rightarrow^{a} s$ and $(g)_s \models \varphi$,
(iii) $g \models \varphi \wedge \psi$	if $g \models \varphi$ and $g \models \psi$,
(iv) $g \models \neg \varphi$	if $g \not\models \varphi$.

We called the satisfaction relation above *strong* because it is tailored to strong bisimulation equivalence. Hennessy and Milner established that for finitely branching process graphs g and h

$$g \leftrightarrow h \Leftrightarrow \forall \varphi \in \mathrm{HML}(g \models \varphi \Leftrightarrow h \models \varphi).$$

In Milner [1985], this characterization was generalized to infinitely branching processes by replacing the finite conjunction $\wedge$ by an infinite conjunction $\bigwedge_{i \in I}$.

This result was adapted to weak bisimulation equivalence by replacing the modality $\langle \tau \rangle$ by $\langle \varepsilon \rangle$ and changing the satisfaction relation a bit. What we call the *weak* notion of satisfaction, $\models_{\mathrm{w}}$, is given by

$g \models_{\mathrm{w}} \langle a \rangle \varphi$	if	there is a path $\mathrm{root}(g) \Rightarrow \rightarrow^{a} \Rightarrow s$ and $(g)_s \models_{\mathrm{w}} \varphi$;
$g \models_{\mathrm{w}} \langle \varepsilon \rangle \varphi$	if	there is a path $\mathrm{root}(g) \Rightarrow s$ and $(g)_s \models_{\mathrm{w}} \varphi$;

and the same clauses for T, $\wedge$ (or $\bigwedge$), and $\neg$ as we had before. Now one obtains:

$$g \leftrightarrow_{\mathrm{w}} h \Leftrightarrow \forall \varphi \in \mathrm{HML}(g \models_{\mathrm{w}} \varphi \Leftrightarrow h \models_{\mathrm{w}} \varphi).$$

Equivalently, one can introduce a modality $\langle \sigma \rangle$ for any sequence σ of visible actions and put

$$g \models_{\mathrm{w}} \langle \sigma \rangle \varphi \quad \text{if} \quad \text{there is a path } \mathrm{root}(g) \Rightarrow^{\sigma} s \text{ and } (g)_s \models_{\mathrm{w}} \varphi.$$

In fact, Hennessy and Milner considered a different kind of observational equivalence than what is now known as weak bisimulation equivalence, and this version of observational equivalence is modally characterized just as weak bisimulation above, but without the modality $\langle \varepsilon \rangle$.

In Milner [1981] a modal characterization of delay bisimulation was given. Define the *delay* notion of satisfaction, $\models_{\mathrm{d}}$, by

$g \models_{\mathrm{d}} \langle a \rangle \varphi$	if	there is a path $\mathrm{root}(g) \Rightarrow \rightarrow^{a} s$ and $(g)_s \models_{\mathrm{d}} \varphi$;
$g \models_{\mathrm{d}} \langle \varepsilon \rangle \varphi$	if	there is a path $\mathrm{root}(g) \Rightarrow s$ and $(g)_s \models_{\mathrm{d}} \varphi$;

and the same clauses for T, $\wedge$ (or $\bigwedge$) and $\neg$ as before. Now one obtains:

$$g \mathrel{\underline{\leftrightarrow}}_{\mathrm{d}} h \Leftrightarrow \forall \varphi \in \mathrm{HML}(g \models_{\mathrm{d}} \varphi \Leftrightarrow h \models_{\mathrm{d}} \varphi).$$

Actually, this is not precisely Milner's characterization, as he simultaneously changed the treatment of divergence.

Now the question arises if a modal characterization can also be obtained for branching equivalence. A positive answer was given in De Nicola and Vaandrager [1990a], but it required the addition of a family of *until*-operators to HML. These operators are also denoted $\langle a \rangle$, but can be distinguished from the operators above because they are binary. $\varphi \langle a \rangle \psi$ says that, during a sequence of internal actions, formula φ remains valid until the action a happens, and afterwards ψ holds:

$g \models_{\mathrm{b}} \varphi \langle a \rangle \psi$ if there is a path $\mathrm{root}(g) = s_0 \rightarrow^{\tau} s_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} s_n \rightarrow^{a} s$ $(n \geq 0)$ such that $(g)_{s_i} \models_{\mathrm{b}} \varphi$ for $i = 0, 1, \ldots, n$ and $(g)_s \models_{\mathrm{b}} \psi$

$g \models_{\mathrm{b}} \varphi \langle \varepsilon \rangle \psi$ if there is a path $\mathrm{root}(g) = s_0 \rightarrow^{\tau} s_1 \rightarrow^{\tau} \cdots \rightarrow^{\tau} s_n$ $(n \geq 0)$ such that $(g)_{s_i} \models_{\mathrm{b}} \varphi$ for $i = 0, 1, \ldots, n-1$ and $(g)_{s_n} \models_{\mathrm{b}} \psi$.

Now the unary modalities $\langle a \rangle \varphi$ can be regarded as abbreviations for $T\langle a \rangle \varphi$. In this sense is the HML with until operators an extension of HML under the delay interpretation. They established that

$$g \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} h \Leftrightarrow \forall \varphi \in \mathrm{HML} + \text{`until'}(g \models_{\mathrm{b}} \varphi \Leftrightarrow h \models_{\mathrm{b}} \varphi).$$

In Van Glabbeek [1993b], a variation of this characterization is proposed. There binary modalities a and τ are considered, written without angular brackets, and defined by

$g \models_{\mathrm{b}} \varphi\, a\, \psi$ if there is a path $\mathrm{root}(g) \Rightarrow^{\tau} r \rightarrow^{a} s$ such that $(g)_r \models_{\mathrm{b}} \varphi$ and $(g)_s \models_{\mathrm{b}} \psi$;

$g \models_{\mathrm{b}} \varphi\, \tau\, \psi$ if there is a path $\mathrm{root}(g) \Rightarrow^{\tau} r$ and $(r = s$ or $r \rightarrow^{\tau} s)$ with $(g)_r \models_{\mathrm{b}} \varphi$ and $(g)_s \models_{\mathrm{b}} \psi$.

Also these modalities, together with T, $\bigwedge$, and $\neg$, characterize branching bisimulation equivalence (using the characterization in terms of semi-branching bisimulations from the proof of Lemma 2.5). This modal language can easily be expressed in the one of De Nicola and Vaandrager. Namely,

$$g \models_{\mathrm{b}} \varphi\, a\, \psi \Leftrightarrow g \models_{\mathrm{b}} T\langle \varepsilon \rangle(\varphi \langle a \rangle \psi) \quad \text{and}$$
$$g \models_{\mathrm{b}} \varphi\, \tau\, \psi \Leftrightarrow g \models_{\mathrm{b}} T\langle \varepsilon \rangle(\varphi \wedge (\varphi \langle \varepsilon \rangle \psi)).$$

The reverse is established in Laroussinie et al. [1995], thus showing that the two languages are equally expressive.

In Van Glabbeek [1993b] also a modal characterization of η-bisimulation is provided. Besides the HML modalities is has a binary modality a (for $a \neq \tau$), as in the characterization above, but this time the interpretation of the modalities is weak, rather than delay-like.

$$g \models_\eta \varphi a \psi \quad \text{if} \quad \text{there is a path root}(g) \Rightarrow^\tau r \rightarrow^a \Rightarrow^\tau s \text{ such that}$$
$$(g)_r \models_\eta \varphi \text{ and } (g)_s \models_\eta \psi.$$

Again $\langle a \rangle \varphi$ can be regarded as an abbreviation for $T a \varphi$. However, any attempt to generalize the modality $\langle \varepsilon \rangle \varphi$ to a binary version yields, under the weak interpretation, a modality that is no more expressive than $\langle \varepsilon \rangle \varphi$. And indeed, HML together with the binary modalities a (for $a \neq \tau$), yields a modal characterization of η-bisimulation:

$$g \underline{\leftrightarrow}_\eta h \Leftrightarrow \forall \varphi \in \mathrm{HML}^\eta (g \models_\eta \varphi \Leftrightarrow h \models_\eta \varphi).$$

Considering the absence of a binary modality for silent activity in the η-characterization, it appears that one variant is still unexplored, namely HML with the binary modalities a (for $a \neq \tau$), or equivalently $\langle a \rangle$ (for $a \neq \tau$), without a binary modality for internal activity, but employing the delay (= branching) interpretation. This possibility is investigated in van Benthem, et al. [1994], and the equivalence characterized by these modalities is called *simple branching bisimulation equivalence*. It happens to be the same as the *quasi-branching bisimulation* independently proposed by Cherief [1992] and discussed in the previous section.

In Milner [1980; 1981] the name observation equivalence is justified by the presentation of a scenario in which two processes are observation inequivalent iff there is an experiment on which one reacts differently than the other. A successful experiment on a process p corresponds with an HML formula satisfied by p, and, vice versa, for each HML formula satisfied by p there is an experiment showing so. In order to upgrade this testing scenario to one that distinguishes branching inequivalent processes by experiment, it suffices to find experiments corresponding to the 'until'-operators added to HML. For this purpose, one may assume that a process leaves a continuous chain of coredumps when it proceeds. Here each coredump contains all information to reconstruct the process in the state were its core was dumped, and *continuous* means that in every state at least one, but possibly many coredumps are left. By examining all coredumps of a process prior to the execution of an action a, it is possible to ascertain that a formula φ holds until a occurs. Similarly one can check that during a series of internal actions φ holds until ψ holds. More details can be found in Van Glabbeek [1993b].

Besides HML with until operators, several other modal characterizations of branching bisimulation have been proposed. First of all each of the logics (A)CTL and (A)CTL* without nexttime operator characterize (a variant of) branching bisimulation [De Nicola and Vaandrager 1990; 1995]. See Argument (5) in the conclusion. These logics are more expressive than HML + 'until'.

In De Nicola et al. [1990b], it has been established that if in the definition of $*$-bisimulation, for $* \in \{\mathrm{w}, \mathrm{b}, \eta, \mathrm{d}\}$, it is required that moves in the one process can be simulated by the other process, not only when going forward but also when going back in history, these modified notions all coincide with branching bisimulation. This yields another modal characterization of branching bisimulation, namely HML with backward modalities.

A last possibility may be adding the eventually operator to HML. It remains to be determined for which classes of process graphs HML + "eventually" is adequate.

10. *Conclusion*

In this paper, we introduced a new semantic equivalence for concurrent systems that we called *branching bisimulation equivalence*. We compared branching bisimulation with the coarser notion of weak bisimulation equivalence and two intermediate notions. Our main motivation for introducing branching bisimulation is that it preserves the branching structure of processes. Although we believe that this has been demonstrated in Sections 2 and 4, this paper does not contain a formal definition of the "branching structure" of a process. Such a definition will be offered in van Glabbeek [1994].

Here we would like to stress that this feature of branching bisimulation is of more than philosophical interest. Actually, we think that of all the equivalences in the linear time-branching time spectrum, branching bisimulation is most suited for verifying the correctness of concurrent systems in applications were the final word on what exactly is observable behaviour has not been pronounced. Below, we list our arguments:

(1) In applications in which the notion of observable behavior is clear, the most suitable equivalence is usually the one which is *fully abstract* with respect to this notion of observable behaviour, that is, identifies two processes if and only if their observable behavior is the same. However, if there is no clarity on what is observable, a verification in a fully abstract semantics with respect to any notion of observability needs to be redone every time one discovers that a little bit more can be observed than what was originally accounted for. Moreover, the soundness of the verification depends crucially on the right estimation of what can be observed. A verification (of the equivalence of two processes) in a semantics that preserves the internal structure of processes, on the other hand, does not depend on considerations of observability (as long as it is clear that no more can be observable than this internal structure), and is automatically valid in any semantics that is fully abstract with respect to some notion of observable behavior. Now for the simple kind of processes that are the subject of this paper, the best formalization of the internal structure of a process appears to be its branching structure. (However, for more complex systems, the internal structure may involve more, for example, the *causal structure* of processes.)

(2) The above argument says that branching bisimulation equivalence (or congruence) is suited for applications in which there is no certainty on what constitutes observable behavior. However, it does not show that it is the only such notion. It could be that there exists a coarser equivalence that still preserves the upperbound on observable behavior. In fact, the name *observation equivalence* suggest that weak (or delay?) bisimulation equivalence is the coarsest equivalence with this property. This argument is enforced by Milner's testing scenario for observation equivalence, presented in Milner [1980; 1981]. However, the testing scenario briefly sketched Section 9 and elaborated in van Glabbeek [1993b] shows (together with argument (1)) that, in fact, branching bisimulation represents the limit of observable behavior, and hence no coarser equivalence shares the advantage mentioned in argument (1).

Of course, one could argue that the testing scenario of Section 9 is not very realistic. However, the same can be said of the testing scenario for weak bisimulation, and the question of what *is* a realistic testing scenario is exactly the one that we want to avoid.

For those readers that believe that some variant of *ready simulation equivalence* represents the limit of observable behavior and is therefore the right notion (cf. Bloom et al [1995] and Ulidowski [1992]), we refer to the *lifeness* argument in van Glabbeek [1993b].

(3) This leaves us with the question whether a finer equivalence than branching bisimulation might be more or equally suitable. In the absence of silent moves, the only candidate appears to be *tree equivalence* [van Glabbeek 1994] as even finer equivalences, such as graph isomorphism, are clearly useless from the point of view of practical applications. But tree semantics has the disadvantage that the standard operational and denotational interpretations of CCS-like system description languages do not coincide. Moreover, the operational interpretation is not compositional, and the most plausible fix requires an upgrade of the underlying graph model into a multigraph model (allowing more than one equally labeled edges between two nodes).

Thus, it is more tempting to search for generalizations of bisimulation equivalence to a setting with silent moves that are finer than branching bisimulation. Such generalizations undoubtedly exists. But in many applications we are interested in three useful properties:

(i) The equivalence is abstract in the sense that it satisfies at least $a\tau x = ax$ (Milner's first τ-law);
(ii) It is a congruence for the operators of CCS and CSP;
(iii) The merge or parallel composition satisfies the expansion theorem of Milner [1980; 1983; 1985; 1989], that is, interleaving semantics is employed, and also the other laws of strong bisimulation are satisfied.

In such circumstances branching bisimulation on finite closed terms is completely axiomatized by the first τ-law ($a\tau x = ax$) and the laws of strong bisimulation, and hence is the finest congruence possible [van Glabbeek 1993a].

Returning to tree equivalence, a similar argument as above applies: All recursion-free closed instances of the law $a(\tau(y+z)+y) = a(y+z)$ are derivable from $a\tau x = ax$ and the laws for (strong) tree equivalence (which are the laws of strong bisimulation without $x + x = x$). Thus, in applications where properties (i)–(iii) above are desired, a "weak" version of tree congruence would be needed, that satisfies $a(\tau(y+z)+y) = a(y+z)$. Such a weak tree-congruence would not have the intuitive appeal of strong tree equivalence, and does not appear to have any advantages over branching bisimulation congruence.

(4) The crucial difference between branching bisimulation and weak bisimulation is that the first one better takes into account the intermediate states of two equivalent processes as they progress through a computation. As argued already by Hennessy and Milner [1980] on the occasion of the introduction of the first version of observation equivalence, it is useful to do so, "because different intermediate states can be exploited in different program contexts to produce different overall behavior." Here, we present an example of a program context that exploits the different intermediate states of two observation

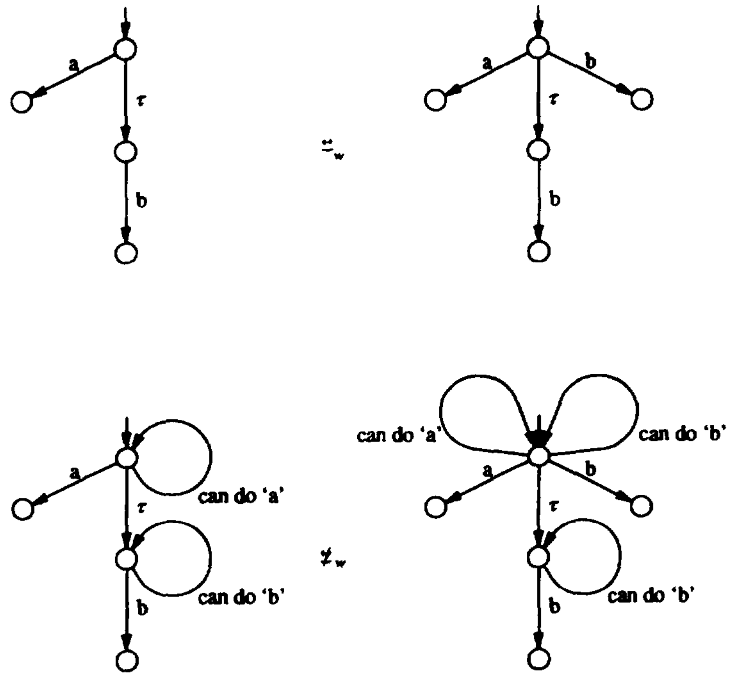

FIG. 7. An operator exploiting different intermediate states of weakly bisimilar processes.

equivalent (i.e., weakly bisimilar) processes to produce different overall behavior. It is a context (operator) that allows a process (its argument) to proceed normally, but in addition can report that the process is ready to perform a visible action, without actually doing it. Thus, the states of this context are the same as the states of its argument, and the transitions are the transitions of its argument, together with a transition labelled "can do 'a' " form a state to itself, whenever the argument can do an "a" from that state. Observe that the processes $a + \tau b$ and $a + \tau b + b$ displayed in Figure 7 are weakly bisimilar. They are even delay bisimilar, but not branching bisimilar. After placing them in the described context (the results are displayed below them) they are no longer weakly bisimilar. In fact, they even have different traces, as only the second one has a trace "can do 'b' " followed by "a."

In Bloom [1995], a class of operators is given for which rooted branching bisimulation equivalence is a congruence. He also presents a class of operators for which rooted weak bisimulation equivalence is a congruence. The latter class is a subclass of the former, and the operator described above falls in the difference. Also all abstract equivalences in the linear time-branching time spectrum that are situated between trace equivalence and delay bisimulation (and including these) fail to be congruences for this operator. On the other hand, we know of no useful operator for which some abstract equivalence in the linear time-branching time spectrum is a congruence, but rooted branching bisimulation is not.

(5) As mentioned in the introduction, weak bisimulation semantics is not adequate for a modal logic with eventually operator. From the examples in the introduction, one can see that the problem originates from the circumstance

that weak bisimulation equivalence does not preserve the branching structure of processes. They also apply to all other semantic equivalences that do not preserve branching, and indeed one can easily prove that such an operator would cause no problems in branching bisimulation semantics, at least not in the variant with explicit divergence. In fact, a much stronger result has been proved by De Nicola and Vaandrager [1995].

The Computation Tree Logic (CTL* [Emerson and Halpern 1986]) is a very powerful logic, combining both branching time and linear time operators. It is a generalization of CTL [Clarke and Emerson 1981], that contains only branching time operators. CTL* is interpreted on Kripke structures (directed graphs of which the nodes are labelled with sets of atomic propositions). De Nicola and Vaandrager [1995] established a translation from process graphs to Kripke structures, so that CTL* can also be regarded as a logic on process graphs. In fact, in De Nicola and Vaandrager [1990], they introduced a counterpart ACTL* of CTL* on process graphs and supplied translations in both directions. One of the operators of CTL/CTL*, the nexttime operator X, makes it possible to see when an (invisible) action takes place, and is therefore incompatible with abstraction. This operator was also criticized by Lamport [1983]. Browne et al. [1988] found that CTL without X and CTL* without X induce the same equivalence on Kripke structures, which they characterized as *stuttering equivalence*. In De Nicola and Vaandrager [1995] branching bisimulation, after being translated to Kripke structures, is shown to coincide with stuttering equivalence. (To be precise, they consider two variants of CTL*, that correspond to two variants of stuttering equivalence and two variants of branching bisimulation, namely *divergence blind branching bisimulation* (our notion with fair abstraction) and *divergence sensitive branching bisimulation* (defined as branching bisimulation with explicit divergence as in Section 8, but also considering endnodes to be divergent). The stuttering equivalence of Browne, et al. [1988] is the divergence sensitive variant.) Hence, (divergence sensitive) branching bisimulation is adequate for CTL* $-$ X. Since the eventually operator of Graf and Sifakis [1987] can be expressed in CTL*, this implies that it causes no problems in branching bisimulation semantics.

(6) The extra identifications made in weak bisimulation semantics on top of branching bisimulation semantics can be cumbersome in certain applications of the theory. An example concerns the work of Jonsson and Parrow [1993], mentioned in the introduction. On the other hand, we are not aware of a single application where weak bisimulation semantics can be successfully applied, but the extra distinctions made in branching bisimulation semantics pose a problem.

In Bouali [1992], an example is given where weak bisimulation semantics works better than branching bisimulation semantics. The example concerns the minimization of Peterson's [1981] mutual exclusion algorithm. Here *minimization* means finding an equivalent process that is as small as possible. In branching bisimulation semantics, this yields a process with 17 states, whereas in weak bisimulation semantics a process with only 14 states is obtained. (In fact, using the quasi-branching bisimulation mentioned in Sections 7 and 8 would bring the number of states to 14 already.) It should be noted however, that weak bisimulation is still far from optimal for this purpose. *Coupled simulation*, proposed by Parrow and Sjödin [1992], is a generalization of bisimulation semantics to a setting with silent moves that is coarser than weak bisimulation. It is completely axiomatized by the laws of weak bisimulation together with $\tau(\tau x + y) = \tau x + y$. Minimization of Peterson's algorithm in coupled simulation semantics would yield a process with no more than 9 states, and using failure semantics [Brookes et al. 1984; De Nicola and Hennessy 1984] would bring it down to 4.

We conjecture that this is illustrative for a general tendency. Coupled simulation has distinct advantages over weak (and branching) bisimulation in applications were the latter notions are too fine. Examples of such applications can be found in Parrow and Sjödin [1992] and van Glabbeek and Vaandrager [1993]. However, whenever weak bisimulation performs better than branching bisimulation, it turns out to be the case that neither of the two notions are really suitable, and coupled simulation, or an even coarser equivalence, is called for.

(7) No abstract semantic equivalence used in concurrency theory is as easy to decide as branching bisimulation congruence. For context-free process without silent actions bisimulation equivalence is the only equivalence in the linear time-branching time spectrum surveyed in Van Glabbeek [1990b] that is decidable at all (cf. Groote and Hüttel [1994] and Christensen et al. [1995]). For finite-state processes, (strong) bisimulation equivalence can be decided in polynomial time, whereas most other equivalences in the linear time-branching time spectrum are PSPACE-complete (cf. Kanellakis and Smolka [1990]).

In Groote and Vaandrager [1990], an algorithm is presented for deciding branching bisimulation equivalence between finite-state processes, with (time) complexity $O(l + n \cdot m)$. Here l is the size of Act, n is the number of nodes in the investigated process graphs, and m the number of edges. The fastest algorithm for weak bisimulation equivalence up till then had time complexity $O(l \cdot n^{2.376})$. In general $n \leq m \leq l \cdot n^2$, so it depends on the density of edges in a graph which algorithm is faster. In a trial implementation of the scheduler of Milner [1980], reported in Groote and Vaandrager [1990], branching bisimulation turned out to be much faster. Furthermore, it turned out that in such automatic verifications the space complexity was a much more serious handicap than the time complexity (the weak bisimulation tools suffered from lack of memory already when applied to processes with 15.000 states). The space complexity of the algorithm of Groote and Vaandrager [1990] is $O(n + m)$, which is less than the space complexity of the weak bisimulation algorithm. Recently, Bouali [1992] proposed another algorithm for weak bisimulation that has the same time complexity as the branching bisimulation algorithm. A trial with Milner's scheduler shows that it is somewhat slower, but with a constant factor only. The space complexity of Bouali's algorithm is $O(n + m^+)$, where m^+ is the number of edges after taking the τ^+ transitive closure, obtained by adding an edge $p \xrightarrow{\tau} q$ whenever there is a nonempty path $p \Rightarrow q$. This improves the space complexity of the old algorithm for weak bisimulation, but is not quite as good as the branching bisimulation algorithm.

(8) For sequential processes, branching bisimulation is preserved under refinement of actions, whereas weak bisimulation is not. This was established in Section 7, which appeared before as [van Glabbeek and Weijland 1989b]. A proof can also be found in Darondeau and Degano [1989].

(9) All $*$-bisimulations ($* \in \{\text{w}, \text{b}, \eta, \text{d}\}$) have relatively simple equational characterizations (see Section 3). However, the listed axioms are in no way self-evident, but arise from the semantic presentation of the respective notions.

In the presence of the (plausible) axioms of strong equivalence, branching bisimulation congruence on finite closed terms is completely axiomatized by the (equally plausible) first τ-law ($a \tau x = ax$) alone [Van Glabbeek 1993a]. This means that the algebra of branching bisimulation can be understood without appealing to semantic notions at all.

(10) The axiom system for branching bisimulation can easily be turned in a complete term rewriting system, which is not the case for the other abstract bisimulation semantics. Work in this direction has been done in Akkerman and Baeten [1991] (in the framework of ACP) and De Nicola et al. [1990a] (in the framework of CCS).

(11) Branching bisimulation equivalence has a nice characterization as weak *back-and-forth bisimulation* (see the end of Section 9). No matter whether the weak, delay, η-, or branching mode is selected, if it is required that moves in the one process can be simulated by the other, not only when going forward but also when going back in history, these modified notions all coincide with branching bisimulation [De Nicola et al. 1990b].

The argument is balanced, however, by a nice characterization of weak bisimulation of which there is no analogy for branching bisimulation. Namely if (root-unwound) process graphs are saturated by adding an edge $s \to^a s'$ whenever there is path $s \Rightarrow \to^a \Rightarrow s'$ (cf. Definition 5.14) and by adding a τ-loop in every state except the root, then two graphs are weakly equivalent iff their saturated graphs are strongly equivalent.

(12) Even if branching bisimulation would not be found interesting in its own right, it sometimes serves as a better tool to prove properties about weak bisimulation than weak bisimulation itself. This has been illustrated by the completeness proof for weak bisimulation in Section 5, which is somewhat shorter than the one in Bergstra and Klop [1985]. A more striking example will be offered in Aceto et al. [1995].

ACKNOWLEDGMENT. We thank the referees for their many insightful suggestions.

REFERENCES

ABRAMSKY, S. 1987. Observation equivalence as a testing equivalence. *Theoret. Comput. Sci. 53*, 225–241.

ACETO, L., FOKKING, W. J., VAN GLABBEEK, R. J., AND INGÓLFSDÓTTIR, A. 1995. Axiomatizing prefix iteration with silent steps. BRICS Research Report RS-95-56. Department of Mathematics and Computer Science. Aalborg Univ., Aalborg, Denmark. (*Inf. Comput.*, to appear.)

AKKERMAN, G. J. AND BAETEN, J. C. M. 1991. Term rewriting analysis in process algebra. *CWI Quarterly 4*, 4, 257–267.

BAETEN, J. C. M., BERGSTRA, J. A. AND KLOP, J. W. 1987a. On the consistency of Koomen's fair abstraction rule. *Theoret. Comput. Sci. 51*, 1/2, 129–176.

BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. 1987b. Ready-trace semantics for concrete process algebra with the priority operator. *Comput. J. 30*, 6, 498–506.

BAETEN, J. C. M. AND VAN GLABBEEK, R. J. 1987. Another look at abstraction in process algebra, In *Proceedings of ICALP 87* (Karlsruhe, Germany). Th. Ottman, ed. Lecture Notes in Computer Science. vol. 267. Springer-Verlag, New York, pp. 84–94.

BAETEN, J. C. M. AND WEIJLAND, W. P. 1990. Process Algebra. In *Cambridge Tracts in Theoretical Computer Science*, vol. 18. Cambridge University Press, Cambridge, England.

BASTEN, T. 1996. Branching bisimulation is an equivalence indeed! *Inf. Proc. Lett.* to appear.

BERGSTRA, J. A. AND KLOP, J. W. 1984. Process algebra for synchronous communication. *Inf. Cont. 60*, 1–3, 109–137.

BERGSTRA, J. A. AND KLOP, J. W. 1985. Algebra of communicating processes with abstraction. *Theoret. Comput. Sci. 37,* 1, 77–121.

BERGSTRA, J. A. AND KLOP, J. W. 1988. A complete inference system for regular processes with silent moves. In *Proceedings of the Logic Colloquium 1986,* F. R. Drake and J. K. Truss, eds. Hull, North Holland, Amsterdam, The Netherlands, pp. 21–81.

BERGSTRA, J. A., KLOP, J. W., AND OLDEROG, E.-R. 1987. Failures without chaos: A new process semantics for fair abstraction. In *Formal Description of Programming Concepts-III. Proceedings of the 3rd IFIP WG 2.2 Working Conference.* M. Wirsing, ed. (Ebberup, Denmark). North Holland, pp. 77–103.

BLOOM, B. 1995. Structural operational semantics for weak bisimulations. *Theoret. Comput. Sci. 146,* 25–68.

BLOOM, B., ISTRAIL, S., AND MEYER, A. R. 1995. Bisimulation can't be traced. *J. ACM 42,* 1 (Jan.), 232–268.

BOUALI, A. 1992. Weak and branching bisimulation in FCTOOL. Rapports de Recherche No. 1575. INRIA, Sophia Antipolis, Valbonne Cedex, France.

BROOKES, S. D., HOARE, C. A. R., AND ROSCOE, A. W. 1984. A theory of communicating sequential processes. *J. ACM 31,* 3 (July), 560–599.

BROWNE, M. C., CLARKE, E. M., AND GRÜMBERG, O. 1988. Characterizing finite Kripke structures in propositional temporal logic. *Theoret. Comput. Sci. 59,* 1/2, 115–131.

CASTELLANO, L., DE MICHELIS, G., AND POMELLO, L. 1987. Concurrency vs Interleaving: An instructive example. *Bull. EATCS 31,* 12–15.

CHERIEF, F. 1992. Contributions à la sémantique du parallélisme: Bisimulations pour le raffinement et le vrai parallélisme. Ph.D. dissertation. Univ. Grenoble, Grenoble, France.

CHERIEF, F. AND SCHNOEBELEN, PH. 1991. τ-Bisimulations and full abstraction for refinement of actions. *Inf. Proc. Lett. 40,* 219–222.

CHRISTENSEN, S., HÜTTEL, H., AND STIRLING, C. 1995. Bisimulation equivalence is decidable for all context-free processes. *Inf. Comput. 121*(2), 143–148.

CLARKE, E. M. AND EMERSON, E. A. 1981. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logics of Programs* D. Kozen, ed. (Yorktown Heights, N.Y.) Lecture Notes in Computer Science, vol 131. Springer-Verlag, New York, pp. 52–71.

DARONDEAU, PH. AND DEGANO, P. 1989. About semantic action refinement. *Fundamenta Informaticae XIV,* 221–234.

DE BAKKER, J. W., BERGSTRA, J. A., KLOP, J. W., AND MEIJER, J.-J. CH. 1984. Linear time and branching time semantics for recursion with merge. *Theoret. Comput. Sci. 34,* 135–156.

DE NICOLA, R. AND HENNESSY, M. 1984. Testing equivalence for processes. *Theoret. Comput. Sci. 34,* 83–133.

DE NICOLA, R., INVERARDI, P., AND NESI, M. 1990a. Using axiomatic presentation of behavioural equivalences for manipulating CCS specifications. In *Automatic Verification Methods for Finite State Systems,* J. Sifakis, ed. Lecture Notes in Computer Science, vol. 407. Springer-Verlag, New York, pp. 54–67.

DE NICOLA, R., MONTANARI, U., AND VAANDRAGER, F. W. 1990b. Back and forth bisimulations. In *Proceedings of CONCUR 90,* J. C. M. Baeten and J. W. Klop, eds. (Amsterdam, The Netherlands, Lecture Notes in Computer Science, vol. 458. Springer-Verlag, New York, pp. 152–165.

DE NICOLA, R. AND VAANDRAGER, F. W. 1990. Action versus state based logics for transition systems. In *Proceedings of the Semantics of Systems of Concurrent Processes,* I. Guessarian, ed. (La Roche Posay, France). Lecture Notes in Computer Science, vol. 469, Springer-Verlag, New York, pp. 407–419.

DE NICOLA, R. AND VAANDRAGER, F. W. 1995. Three logics for branching bisimulation. *J. ACM 42,* 2, 458–487.

DEVILLERS, R. 1992. Maximality preserving bisimulation. *Theoret. Comput. Sci. 102,* 165–183.

EMERSON, E. A. AND HALPERN, J. Y. 1986. 'Sometimes' and 'Not Never' revisited: on branching time versus linear time temporal logic. *J. ACM 33,* 1 (Jan.) 151–178.

GRAF, S. AND SIFAKIS, J. 1987. Readiness semantics for regular processes with silent actions. In *Proceedings of ICALP 87,* Th. Ottman, ed. (Karlsruhe, Germany) Lecture Notes in Computer Science, vol. 267. Springer-Verlag, New York, pp. 115–125.

GROOTE, J. F. AND HÜTTEL, H. 1994. Undecidable equivalences for basic process algebra. *Inf. Comput. 115,* 2, 354–371.

GROOTE, J. F. AND VAANDRAGER, F. W. 1990. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proceedings of ICALP 90.* M. S. Paterson, ed. (Warwick, R. I.). Lecture Notes in Computer Science, vol. 443, Springer-Verlag, New York, pp. 626–638.

HENNESSY, M. AND MILNER, R. 1980. On observing nondeterminism and concurrency. In *Proceedings ICALP 80,* J. W. de Bakker and J. van Leeuwen, eds. Lecture Notes in Computer Science, vol. 85. Springer-Verlag, New York, 299–309 (a preliminary version of Hennessy and Milner [1985]).

HENNESSY, M. AND MILNER, R. 1985. Algebraic laws for nondeterminism and concurrency. *J. ACM 32,* 1, 137–161.

HENNESSY, M. AND PLOTKIN, G. D. 1980. A term model for CCS. In *Proceedings of MFCS 80* (P. Dembiński, ed.). Lecture Notes in Computer Science, vol. 88. Springer-Verlag, New York, pp. 261–274.

HOARE, C. A. R. 1980. Communicating sequential processes. In *On the construction of programs,* R. McKeag and A. M. Macnaghten, eds. Cambridge University Press, Cambridge, England, pp. 229–254.

HOARE, C. A. R. 1985. *Communicating sequential processes.* Prentice-Hall, London, England.

JONSSON, B. AND PARROW, J. 1993. Deciding bisimulation equivalences for a class of non-finite-state programs. *Inf. Comput. 107,* 272–302.

KANELLAKIS, P. C. AND SMOLKA, S. A. 1990. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput. 86,* 43–68.

LAMPORT, L. 1983. What good is temporal logic? *Inf. Proc. 83,* 657–668.

LAMPORT, L. 1986. On interprocess communication. Part 1: Basic formalism. *Dist. Comput. 1,* 2, 77–85.

LAROUSSINIE, F., PINCHINAT, S., AND SCHNOEBELEN, PH. 1995. Translations between modal logics of reactive systems. *Theoret. Comput. Sci. 140,* 1, 53–71.

MILNER, R. 1980. A calculus of communication systems. In *Lecture Notes in Computer Science,* vol. 92. Springer-Verlag, New York.

MILNER, R. 1981. A modal characterisation of observable machine-behaviour. In *Proceedings CAAP 81,* G. Astesiano and C. Böhm, eds. Lecture Notes in Computer Science, vol. 112. Springer-Verlag, New York, pp. 25–34.

MILNER, R. 1983. Calculi for synchrony and asynchrony. *Theoret. Comput. Sci. 25,* 267–310.

MILNER, R. 1985. Lectures on a calculus for communicating systems. In *Proceedings of the Seminar on Concurrency,* S. D. Brookes, A. W. Roscoe, and G. Winskel, eds. Lecture Notes in Computer Science, vol. 97. Springer-Verlag, New York, pp. 197–220.

MILNER, R. 1989. *Communication and concurrency.* Prentice Hall, London, England.

OLDEROG, E.-R. AND HOARE, C. A. R. 1986. Specification-oriented semantics for communicating processes. *Acta Inf. 23,* 9–66.

PARK, D. M. R. 1981. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI Conference* P. Deussen, ed. Lecture Notes in Computer Science, vol. 104. Springer-Verlag, New York, pp. 167–183.

PARROW, J. AND SJÖDIN, P. 1992. Multiway synchronization verified with coupled simulation. In *Proceedings of CONCUR '92* W. R. Cleaveland, ed. (Stony Brook, NY). Lecture Notes in Computer Science, vol. 630. Springer-Verlag, New York, pp. 518–533.

PETERSON, G. L. 1981. Myths about the mutual exclusion problem. *Inf. Proc. Lett. 12,* 3, 115–116.

PHILLIPS, I. C. C. 1987. Refusal testing. *Theoret. Comput. Sci. 50,* 241–284.

PNUELI, A. 1985. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of the 12th ICALP,* W. Brauer, ed. (Nafplion, Greece). Lecture Notes in Computer Science, vol. 194. Springer-Verlag, New York, pp. 15–32.

POMELLO, L. 1986. Some equivalence notions for concurrent systems. An overview. In *Advances in Petri Nets 1985* G. Rozenberg, ed. Lecture Notes in Computer Science, vol. 222. Springer-Verlag, New York, pp. 381–400.

PRATT, V. R. 1986. Modeling concurrency with partial orders. *Int. J. Para. Prog. 15,* 1, 33–71.

ROUND, W. C. AND BROOKES, S. D. 1981. Possible futures, acceptances, refusals and communicating processes. In *Proceedings 22nd Annual Symposium on Foundations of Computer Science* (Nashville, Tenn.). IEEE, New York, pp. 140–149.

ULIDOWSKI, I. 1992. Equivalences on observable processes. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science (LICS '92)* (Santa Cruz, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 148–159.

VAN BENTHEM, J., VAN EIJCK, J., AND STEBLETSOVA, V. 1994. Modal logic, transition systems and processes. *J. Logic Comput. 4*, 5, 811–855.

VAN GLABBEEK, R. J. 1990a. Comparative concurrency semantics and refinement of actions. Ph.D. dissertation. Free University, Amsterdam, The Netherlands.

VAN GLABBEEK, R. J. 1990b. The linear time-branching time spectrum. In *Proceedings of CONCUR '90* J. C. M. Baeten and J. W. Klop, eds. (Amsterdam, The Netherlands) Lecture Notes in Computer Science, vol. 458. Springer-Verlag, New York, pp. 278–297.

VAN GLABBEEK, R. J. 1993a. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proc. MFCS 1993,* A. M. Borzyszkowski and S. Sokolowski, eds. (Gdansk, Poland). Lecture Notes in Computer Science, vol. 711. Springer-Verlag, New York, pp. 473–484.

VAN GLABBEEK, R. J. 1993b. The linear time–branching time spectrum II. Preliminary version available from Boole.stanford.edu; extended abstract in *Proceedings of CONCUR '93.* E. Best, ed. (Hildesheim, Germany). Lecture Notes in Computer Science, vol. 715. Springer-Verlag, New York, pp. 66–81.

VAN GLABBEEK, R. J. 1994. What is branching time and why to use it? Report No. STAN-CS-93-1486. Stanford University, available from Boole.stanford.edu. In *the Concurrency Column,* M. Nielsen, Ed. *Bull. EATCS 53,* 190–198.

VAN GLABBEEK, R. J. AND GOLTZ, U. 1990. Refinement of actions in causality based models. In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness* J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, eds. (Mook, The Netherlands), Lecture Notes in Computer Science, vol. 430. Springer-Verlag, New York, pp. 267–300.

VAN GLABBEEK, R. J. AND VAANDRAGER, F. W. 1993. Modular specification of process algebras. *Theoret. Comput. Sci.*, vol. 113, 2, 293–348.

VAN GLABBEEK, R. J. AND WEIJLAND, W. P. 1989a. Branching time and abstraction in bisimulation semantics (extended abstract), Information Processing 89. In *Proceedings of the IFIP 11th World Computer Congress,* G. X. Ritter, ed. (San Francisco, Calif.). North Holland, Amsterdam, The Netherlands, pp. 613–618.

VAN GLABBEEK, R. J. AND WEIJLAND, W. P. 1989b. Refinement in branching time semantics. Rep. CS-R8922. Centrum voor Wiskunde en Informatica, Amsterdam, and *Proceedings of AMAST Conference.* Univ. Iowa, Iowa City, Ia. pp. 197–201.

VAN GLABBEEK, R. J. AND WEIJLAND, W. P. 1991. Branching time and abstraction in bisimulation semantics. Rep. CS-R9120. CWI, Amsterdam, The Netherlands.

WALKER, D. J. 1990. Bisimulation and divergence. *Inf. Comput. 85,* 2, 202–241.

WEIJLAND, W. P. 1989. Synchrony and asynchrony in process algebra. Ph.D. dissertation. Univ. Amsterdam, The Netherlands.

RECEIVED AUGUST 1991; REVISED JULY 1995; ACCEPTED OCTOBER 1995