Branching bisimulation as a strong bisimulation

A. Bouali, R. De Nicola

Computer Science/Department of Software Technology

Report CS-R9452 September 1994

Branching Bisimulation as a Strong Bisimulation

Amar Bouali
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
`amar@cwi.nl`

Rocco De Nicola
Università di Roma "La Sapienza"
Via Salaria, 113, I-00198 Roma, Italy
`denicola@vm.cnuce.cnr.it`

Abstract

An alternative definition of *branching bisimulation* as a strong equivalence is proposed. It permits taking advantage of known efficient proof techniques for checking equivalence of finite labelled graphs and throws light on the nature of branching bisimulation equivalence. Branching bisimilarity of two labelled graphs with silent moves is established by considering their corresponding rigid (without silent moves) graphs with labels on both arcs and nodes (these are labelled by families of rigid graphs) and proving their strong equivalence. An adaptation of Paige and Tarjan partition algorithm leads to the lowest known time complexity algorithm for checking branching bisimulation.

AMS Subject Classification (1991): 68N99, 68Q10, 68Q20, 68Q25, 68Q60.
CR Subject Classification (1991): F.1.1, F.2.2, F.3.2.
Keywords & Phrases: concurrency semantics, observational bisimulation equivalences, rigid bisimulation equivalence, labelled transition systems, bisimulation algorithm, relational coarsest partition problem.
Note: The first author started this work while visiting the *Istituto di Elaborazione dell'Informazione* of CNR in Pisa and was funded by the European Research Consortium for Informatics and Mathematics, fellowship programme. The second author was funded by the LAMBRUSCO project, supported by the CNR Finalized Project for Information Systems and Parallel Computation, Contr. N. 93.00894.69 and by *Istituto di Elaborazione dell'Informazione* of CNR in Pisa.

1. Introduction

Behavioural equivalences play an important rôle in the description of the operational semantics of concurrent systems. These equivalences are means for abstracting from the irrelevant details introduced when describing systems as sets of states that evolve by performing actions, i.e. by means of labelled transition systems. There are various opinions about which features of a system are relevant for a given purpose, and hence various notions of equivalence for labelled transition systems have been proposed: [4], [6] and [7] give comparative accounts.

One of the best known behavioural equivalence is *observational equivalence* [11]; it leads to considering two systems as equivalent whenever they can perform the same sequence of actions to reach observationally equivalent states. Observational equivalence is called *strong* when all labels of transitions are considered as visible and *weak* when some (internal, invisible) actions are ignored. Both these equivalences rely on the existence of a bisimulation that contains equivalent processes. Bisimulations are binary relations over dynamic systems that are closed under transitions, in the sense that they contain only pairs of systems that can evolve to bisimilar pairs via equal transitions [13]. The notion of

Report CS-R9452
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

bisimulation has proved of fundamental importance for working with structures used to describe non-deterministic systems. It leads to structural equivalences that naturally take the branching structure of systems into account and, due to the associated proof techniques, can be efficiently verified.

Strong observational equivalence has been generalized to systems with silent moves also in other ways. One of its most popular alternative generalizations is *branching bisimulation equivalence* [8]. This considers two systems as equivalent only if every computation, i.e. every sequence of (visible and silent) actions and states, of one system has a correspondent in the other; corresponding computations have the same sequence of visible actions and are such that all their intermediate states have equivalent potentials. Now, both weak and branching bisimulation equivalence ignore τ-actions, but they treat intermediate states accessed via τ transitions differently: branching bisimulation requires that all intermediate states be equivalent to either the start or the finish state of the weak visible transition, whereas weak bisimulation does not constrain them. These two equivalences, then, lead to different identifications, putting a different stress on the branching structure of processes; with branching bisimulation being more demanding than weak observational equivalence.

If one wants to use process algebras as verification tools, proving efficiently (strong, weak, branching) bisimulation equivalence of transition systems is of fundamental importance. Indeed, one can use process algebras terms to provide different descriptions of a given system and establish correctness by proving equivalence of the descriptions under certain observational assumptions. Efficient algorithms have been proposed for checking strong bisimulation equivalence that are based on the partition refinement algorithm known as *Relational Coarsest Partition* (RCP), [10, 12]. The efficientest one is of Paige and Tarjan and it has an $\mathcal{O}(m.\log n)$ time complexity where m is the number of transitions and n the number of states. The very same algorithm can be exploited to prove weak observational equivalence by first using a transitive closure algorithm to compute a new transition relation that "by-passes" silent transitions and then applying the RCP algorithm to the new transition system without silent transition that we call *rigid*. The most efficient known algorithm for *branching bisimulation* is that of [9], as an adaption of the Kanellakis and Smolka algorithm, and has a $\mathcal{O}(m.n)$ time complexity.

In this paper we propose an alternative definition of *branching bisimulation* as a strong observational equivalence over rigid labelled transition systems whose states are decorated by additional information (sets of rigid graphs). This characterization will enable us to use a variant of Paige and Tarjan algorithm to check branching bisimulation and to offer lowest time complexity algorithm for this. Branching bisimilarity of two labelled graphs is established by considering their corresponding rigid graphs with labels on both arcs and nodes (these are labelled by families of rigid graphs) and proving their strong equivalence.

The alternative characterization has also the effect of throwing additional light on the nature of branching bisimulation and of showing how by varying the type of information associated to the states and the requirements on it new equivalences tailored to the specific application can be obtained.

2. Basic Definitions

We start by setting standard definitions and notations for labelled transition systems and bisimulation equivalences. We will mainly rely on the notational conventions of [5].

2.1 Labelled Transitions Systems

We consider a set of labels A of *visible* actions ranged over by $a, b, \ldots$ We let τ be the distinct *internal* action not belonging to A and write A_τ for $A \cup \{\tau\}$, this will be ranged over by $\mu, \nu, \ldots$ Moreover we write A_ϵ for $A \cup \{\epsilon\}$, and use $\iota, \kappa, \ldots$ to range over it. For any set L, we use $L^\star$ to indicate the set of finite sequences over it; $\mathbb{1}$ denotes the empty string; L^+ is the set $L^\star$ without the empty string.

Definition 2.1 *A (finite) labelled transition system (LTS) is a structure* $S =< Q, A_\tau, q_0, \longrightarrow >$, *where Q is a set of* states, *$q_0 \in Q$ is the initial state and* $\longrightarrow \in (Q \times A_\tau \times Q)$

is a (finite) set of (labelled) transitions.

As usual we write $q \xrightarrow{\mu} q'$ *for* $(q,\mu,q') \in \longrightarrow$. *We define the following projections over transitions:* $\alpha(q,\mu,q') = q, \beta(q,\mu,q') = q', \lambda(q,\mu,q') = \mu$. *We also use the following abbreviations:*

$$out_\mu(q) = \{q' \mid q \xrightarrow{\mu} q'\}, out(q) = \bigcup_{\mu \in A_\tau} out_\mu(q), in_\mu(q) = \{q' \mid q' \xrightarrow{\mu} q\}, in(q) = \bigcup_{\mu \in A_\tau} in_\mu(q)$$

An lts is called rigid *if it has no τ-transition. It is called* τ-clean *if it has no τ-cycle.*

Notation 2.1 *We note* $\Longrightarrow$ *the relation defined as:*

- $p \overset{\epsilon}{\Longrightarrow} q$ *iff* $\exists p_0, \ldots, p_n, n \geq 0, p = p_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} p_n = q$.
- $\forall a \in A, p \overset{a}{\Longrightarrow} q$ *iff* $\exists p_1, p_2, p \overset{\epsilon}{\Longrightarrow} p_1 \xrightarrow{a} p_2 \overset{\epsilon}{\Longrightarrow} q$.

For $s = s_1 \ldots s_n \in A_\tau^\star$ *(resp.* $A_\epsilon^\star$*),* $\xrightarrow{s} = \xrightarrow{s_1} \ldots \xrightarrow{s_n}$ *(resp.* $\overset{s}{\Longrightarrow} = \overset{s_1}{\Longrightarrow} \ldots \overset{s_n}{\Longrightarrow}$*).*

In the following, we work with a fixed lts $S = < Q, A_\tau, q_0, \longrightarrow >$, except when differently stated. We shall often use the notion of *runs* that we formally define as:

Definition 2.2 A sequence $(p_0, a_0, p_1) \cdots (p_{n-1}, a_{n-1}, p_n) \in \longrightarrow^\star$ is called a path from p_0. A run from a state $p \in Q$ is a pair (p, π) where π is a path from p. We write $runs_S(p)$ as the set of runs in S starting from p and $runs_S$ the set of all runs in S, ranged over by $\rho, \sigma, \ldots$ We extend projections of definition 2.1 to runs and introduce two new ones, as: for $\rho = (p, (p_0, a_0, p_1) \cdots (p_{n-1}, a_{n-1}, p_n))$ we write $\alpha(\rho) = p_0, \beta(\rho) = p_n$, and if $n > 0$, $\lambda(\rho) = a_1 \cdots a_n$, $begin(\rho) = (p_0, a_0, p_1)$, $end(\rho) = (p_{n-1}, a_{n-1}, p_n)$. Also $\Lambda(\rho)$ as the string obtained from $\lambda(\rho)$ where all the τ's are removed, and $\Lambda_\epsilon(\rho)$ as the string obtained from $\lambda(\rho)$ where each non-empty sequence of τ's is replaced by a single ϵ. Moreover, *states*(ρ) denotes the set of states appearing in ρ. Concatenation of runs is obtained by juxtaposition: $\rho\rho'$ is defined if and only if $\beta(\rho) = \alpha(\rho')$. We extend the transition relations defined on states to runs, where $s \in A^\star$ as:

$$\begin{aligned} \rho_1 \xrightarrow{s} \rho_2 &\iff \exists \rho_1', \lambda(\rho_1') = s \wedge \rho_2 = \rho_1\rho_1' \\ \rho_1 \overset{s}{\Longrightarrow} \rho_2 &\iff \exists \rho_1', \Lambda(\rho_1') = s \wedge \rho_2 = \rho_1\rho_1' \end{aligned}$$

2.2 Bisimulation Equivalences over LTS

Definition 2.3 *A binary relation* $\mathcal{R} \in Q \times Q$ *is a bisimulation if and only if the following holds:*

$$\textit{if } p\mathcal{R}q \textit{ then } \begin{cases} \textit{if } p \xrightarrow{\mu} p' \textit{ then } \exists q' : (q \xrightarrow{\mu} q') \wedge (p'\mathcal{R}q') \\ \textit{if } q \xrightarrow{\mu} q' \textit{ then } \exists p' : (p \xrightarrow{\mu} p') \wedge (p'\mathcal{R}q') \end{cases}$$

We write $\sim$ *for the largest bisimulation relation satisfying the conditions above. We call this strong bisimulation to distinguish it from its weak variant* ($\approx$) *that relies on relation* $\Longrightarrow$.

Definition 2.4 *A binary relation* $\mathcal{R} \in Q \times Q$ *is a branching bisimulation if and only if the following holds:*

$$\textit{if } p\mathcal{R}q \textit{ then } \begin{cases} \textit{if } p \xrightarrow{\mu} p' \textit{ then } \begin{cases} \textit{either } \mu = \tau \wedge p'\mathcal{R}q \\ \textit{or } \exists q_1, q' : (q \overset{\epsilon}{\Longrightarrow} q_1 \xrightarrow{\mu} q'), p\mathcal{R}q_1 \wedge (p'\mathcal{R}q') \end{cases} \\ \textit{if } q \xrightarrow{\mu} q' \textit{ then } \begin{cases} \textit{either } \mu = \tau \wedge p\mathcal{R}q' \\ \textit{or } \exists p_1, p' : (p \overset{\epsilon}{\Longrightarrow} p_1 \xrightarrow{\mu} p'), p_1\mathcal{R}q \wedge (p'\mathcal{R}q') \end{cases} \end{cases}$$

We write $\sim_b$ *for the largest branching bisimulation satisfying the conditions above.*

As usual, two lts $S_1 =< Q_1, A_\tau, q_1, \longrightarrow_1>, S_2 =< Q_2, A_\tau, q_2, \longrightarrow_2>$ are strong (resp. weak, branching) bisimilar, noted $S_1 \sim S_2$ (resp. $S_1 \approx S_2$, $S_1 \sim_b S_2$) if and only if there exists a strong (resp. weak, branching) bisimulation over the lts obtained by the the disjoint union of the sets of states Q_1, Q_2 and the sets of transitions $\longrightarrow_1, \longrightarrow_2$, relating q_1, q_2.
Now we will show that strong and branching bisimulation are closely related by defining the latter strictly in terms of the former. This shall be achieved by studying the rôle of τ actions and capturing it in the new structures. The basic idea is that of collecting together all those states that can be reached from a node via τ-steps and use this information when deciding states equivalence. We shall formalize this by first introducing a graph transformation that, on one hand, "forgets" the τ's, (*rigidity transformation*), on the other, *decorates* states with the set of states connected with them via silent transitions. We shall then define an equivalence over transformed-decorated graphs that only requires strong bisimulation.

3. A Strong Characterization of Branching Bisimulation

3.1 The Internal Non Determinism

Branching bisimulation puts significant stress on the nondeterminism originated by the presence of τ-moves. Here, we set up a formal framework to collect the information about this internal nondeterminism. We only consider LTS that do not have τ-cycles (τ-clean). This is not restrictive, because any lts has a branching bisimulation equivalent one that is τ-clean.

Definition 3.1 *We define two projections left, right on runs, as:*

$$left(\rho) = \begin{cases} \rho, \ \text{if } \Lambda(\rho) = \mathbb{1} \\ \rho_l, \ \text{such that } \rho = \rho_l\rho', \Lambda(\rho_l) = \mathbb{1} \wedge \lambda(begin(\rho')) \neq \tau \ \text{otherwise.} \end{cases}$$

$$right(\rho) = \begin{cases} \rho, \ \text{if } \Lambda(\rho) = \mathbb{1} \\ \rho_r, \ \text{such that } \rho = \rho'\rho_r, \Lambda(\rho_r) = \mathbb{1} \wedge \lambda(end(\rho')) \neq \tau \ \text{otherwise.} \end{cases}$$

Definition 3.2 *Let $p \in Q$. For all sequence $s \in A_\epsilon^\star$, we write $w(s)$ for the string obtained from s where all the ϵ are dropped out. We define the following sets:*

$$\begin{aligned} obs(p,s) &= \{\rho \in run(p), \Lambda(\rho) = w(s)\} \\ min(p,s) &= \{\rho \in obs(p,s) \mid \lambda(end(\rho)) \neq \tau\} \\ max(p,s) &= \{\rho \in obs(p,s) \mid out_\tau(\beta(\rho)) = \emptyset\} \end{aligned}$$

Literally, $min(p,s)$ is the set of runs from p labelled by s such that their last transition is not a τ-transition. Dually, $max(p,s)$ is the set of runs from p labelled by s such that, from their last state a τ-transition can not be performed.

We define here some sets that carry structural information on the branching structure of LTS especially that in correspondence of internal moves.

Definition 3.3 *Let $s \in A_\epsilon^\star$.*

$$\begin{aligned} after(\rho) &= states(right(\rho)) \\ before(\rho) &= states(left(\rho)) \\ Post(p,s) &= \{after(\rho) \mid \rho \in max(p,s)\} \\ Pre(p,s) &= \{before(\rho) \mid \rho \in max(p,s)\} \end{aligned}$$

An example should help us in clarifying these notions.

Example 3.1 *Take graph G_2 displayed on figure 0.1. We have the following:*

$$\begin{array}{ll} Post(0,a) = \{\{1,4\}\} & Post(1,a) = \emptyset \\ Pre(1,e) = \{\{1,4\},\{1\}\} & Pre(0,e) = \emptyset \end{array}$$

Notation 3.1 *Let $R \subseteq Q \times Q$ be any equivaleence relation over Q. $[q]_R$ denotes the equivalence classe of q in R. For $T \in 2^Q$ and $\Theta \in 2^{2^Q}$, we write*

$$\begin{aligned}
[T]_R &= \{[q]_R \mid q \in T\} \\
[\Theta]_R &= \{[T]_R \mid T \in \Theta\} \\
\overline{[T]_R} &= \bigcup_{T \in \Theta} [T]_R
\end{aligned}$$

The following definition permits relating sets of states and sets of sets of states up to some equivalence relation R on them.

Definition 3.4 *Let Θ_1, Θ_2 be sets of states of Q. Let R be any (binary) equivalence relation over $Q \times Q$. We define the ordering relation $\leq_R$ up to R between these sets of states as*

$$\Theta_1 \leq_R \Theta_2 \iff \forall q_1 \in \Theta_1, \exists q_2 \in \Theta_2, q_1 R q_2.$$

This order leads naturally to an equivalence relation that we denote by $\cong_R$:

$$\Theta_1 \cong_R \Theta_2 \iff \Theta_1 \leq_R \Theta_2 \wedge \Theta_1 \geq_R \Theta_2$$

This relation is extended to sets of sets of states as follow:

Definition 3.5 *Let $\mathcal{T}_1, \mathcal{T}_2$ be sets of sets of states. We define the relation $\equiv_R$ up to R between sets of sets of states as*

$$\mathcal{T}_1 \sqsubseteq_R \mathcal{T}_2 \iff \forall \Theta_1 \in \mathcal{T}_1, \exists \Theta_2 \in \mathcal{T}_2, \Theta_1 \cong_R \Theta_2$$

The underlying equivalence denoted as $\equiv_R$ is given by:

$$\mathcal{T}_1 \equiv_R \mathcal{T}_2 \iff \mathcal{T}_1 \sqsubseteq_R \mathcal{T}_2 \wedge \mathcal{T}_1 \sqsupseteq_R \mathcal{T}_2$$

The sets of sets of states we shall consider are the *Pre/Post* sets that we have introduced before. The comparison of *Pre/Post* sets is the conditions that branching bisimulation requires adding to the weak bisimulation scheme. We want to stress that these conditions are somewhat related to the backward conditions required in the weak back and forth variant of bisimulation introduced in [5].

3.2 From LTS to Rigid LTS
We now define a transformation that starting from any lts yields a decorated rigid lts abstracting the transition relation, while forgetting the internal moves and adding *Pre/Post* sets information to each state.

Definition 3.6 *We call $rig_\psi(S) =< Q_r, A, q_{r0}, \longrightarrow_r, Pre, Post >$ the rigid lts (up to isomorphism) obtained from S such that:*

- *There exists a bijection $\psi : Q \rightarrow Q_r$ where:*
  - $\psi(q_{r0}) = q_0$,
  - $q \xrightarrow{a}_r q' \iff (\psi^{-1}(q) \overset{a}{\Longrightarrow} \psi^{-1}(q'))$
- *$Pre, Post : Q \times A_\epsilon \longrightarrow 2^{2^Q}$ are the Pre/Post sets defined above.*

By abuse of notation $rig(q), q \in Q$ stands for the rigid lts corresponding to the lts whose initial state is q.

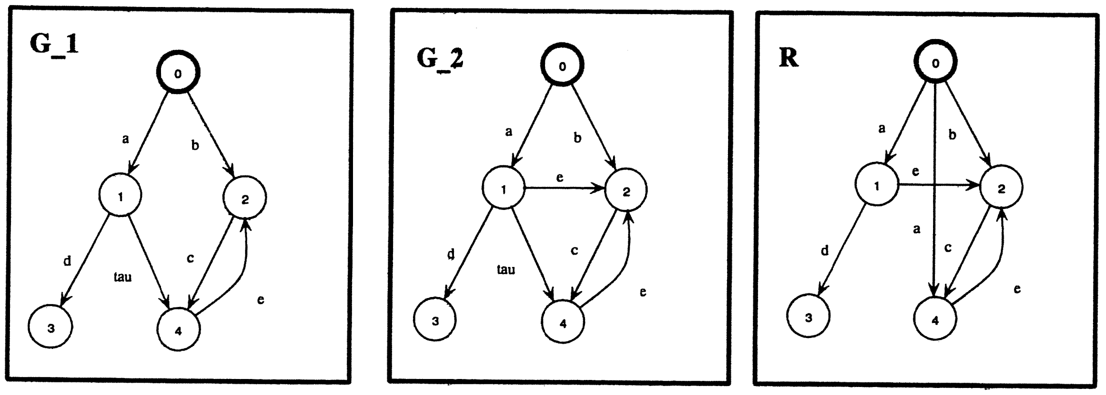

Figure 0.1: Rigidity transformation

Example 3.2 *Graphs displayed on figure 0.1 are examples of rigidity transformation. The two graphs G_1 and G_2 have the same rigid graph called R.*

Here we propose a first result about the rigidity transformation and weak bisimulation equivalence.

Proposition 3.1 *Let p, q be two states of Q and $rig_\psi(S)$ its rigid lts. We have:*

$$p \sim_b q \Rightarrow p \approx q \Rightarrow \psi(p) \sim \psi(q)$$

Proof: $p \sim_b q \Rightarrow p \approx q$ is well known. Let R_w be the maximal weak bisimulation over S and consider the following relation over $Q_r \times Q_r$:

$$R = \{(\psi(p), \psi(q)) \mid (p, q) \in R_w\}$$

We claim that R is a strong bisimulation. Indeed, let $(p, q) \in R$, and let $p \xrightarrow{a}_r p'$ for some $a \in A$ and some $p' \in Q_r$; if we let $\bar{p} = \psi^{-1}(p), \bar{q} = \psi^{-1}(q), \bar{p}' = \psi^{-1}(p')$. By defnition, we have then $\bar{p} \overset{a}{\Longrightarrow} \bar{p}'$. Since $(p, q) \in R$, then $(\bar{p}, \bar{q}) \in R_w$. So, we have $\exists \bar{q}', \bar{q} \overset{a}{\Longrightarrow} \bar{q}' \wedge (\bar{p}', \bar{q}') \in R_w$. Thus if we let $q' = \psi(\bar{q}')$, then $q \xrightarrow{a}_r q'$ and $(p', q') \in R$. □

The converse is naturally false. A counterexample is displayed on figure 0.1, where the two graphs are not branching bisimilar but they have the same rigid graph.
This property gives us a *sufficient* condition on branching bisimilar states; this as a direct consequence on the performed visible action of two equivalent states. We need to relate all the silently visited rigid sub-LTS before a visible action is performed. This shall be more clear in the following where we introduce definitions and notations to help formalization.
We propose an equivalence over rigid LTS, based on strong bisimulation enriched with the extra comparisons of *Pre/Post* sets and prove that it coincides with branching bisimulation. We call it r-bisimulation (for rigid bisimulation).

3.3 The Rigid Bisimulation

Definition 3.7 *A (binary) relation $\phi \subseteq Q \times Q$ is an r-bisimulation if and only if:
If $p\phi q$ then*

1. $\forall a \in A_\epsilon, (Post(p, a) \equiv_\phi Post(q, a) \wedge Pre(p, a) \equiv_\phi Pre(q, a))$
2. $\forall a \in A, (((p \overset{a}{\Longrightarrow} p') \Rightarrow (\exists q', q \overset{a}{\Longrightarrow} q' \wedge p'\phi q')) \wedge ((q \overset{a}{\Longrightarrow} q') \Rightarrow (\exists p', p \overset{a}{\Longrightarrow} p' \wedge p'\phi q')))$

As usual, we denote by $\approxeq$ the largest r-bisimulation. Also, two LTS S_1, S_2 are r-bisimilar, denoted $S_1 \approxeq S_2$ exactly when there exists an r-bisimulation relating their initial states.

Remark 3.1 *We have $Pre(p, \epsilon) = Post(p, \epsilon)$, since for any run ρ such that $\Lambda(\rho) = \mathbb{1}$, $before(\rho) = after(\rho)$.*

The rigid bisimulation is defined over decorated rigid LTS. The first condition relates $Pre/Post$ sets, and considers all action labels, ϵ included. The second condition considers only visible actions when comparing state behaviors. The following proposition relates $\approxeq$ to weak bisimulation; the result is somewhat expected because the new definition has additional requirements on the τ-connected states (internal nondeterminism).

Proposition 3.2

$$(p \approxeq q) \Rightarrow (p \approx q)$$

Proof: We just have to show that

$$(p \overset{\epsilon}{\Longrightarrow} p') \Rightarrow (\exists q', q \overset{\epsilon}{\Longrightarrow} q' \wedge p' \approxeq q')$$

Suppose $\exists p', p \overset{\epsilon}{\Longrightarrow} p'$ such that $\forall q', q \overset{\epsilon}{\Longrightarrow} q'$ we have $p' \not\approxeq q'$. This implies that $Post(p, \epsilon) \not\equiv Post(q, \epsilon)$, contradicting the hypothesis $p \approxeq q$. □

However, $\approxeq$ and $\approx$ are not equal, as shown by the example of figure 0.1, where $G_1 \approx G_2$ but $G_1 \not\approxeq G_2$.

Lemma 3.1

$$p \sim_b q \Rightarrow (\forall a \in A_\epsilon, (Post(p, a) \equiv_{\sim_b} Post(q, a) \wedge Pre(p, a) \equiv_{\sim_b} Pre(q, a)))$$

Proof: We prove this property by construction, showing that when $p \sim_b q$, then for any action $a \in A_\epsilon$, for any $\rho \in obs(p, a)$ there exists $\sigma \in obs(q, a)$ such that $f(\rho) \cong_{\sim_b} f(\sigma)$, for $f \in \{before, after\}$. As we shall prove at the same time the symmetric case because $\sim_b$ is symmetric, then the property shall be proved.
Let $a \in A_\epsilon$ and $\rho \in obs(p, a)$ such that

$$\rho = (p, (p = p_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} p_{n_p} \xrightarrow{a} p'_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} p'_{m_p}))$$

As $p \sim_b q$, we know that $\exists \sigma \in obs(q, a)$ such that:

1. $\rho = (q, (q = q_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} q_{n_q} \xrightarrow{a} q'_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} q'_{m_q}))$
2. $p_0 \sim_b q_0, p_{n_p} \sim_b q_{n_q}, \forall i, 0 < i < n_p, \exists j, 0 \leq j \leq n_q, p_i \sim_b q_j$
3. $p'_0 \sim_b q'_0, p'_{m_p} \sim_b q'_{m_q}, \forall i, 0 < i < m_p, \exists j, 0 \leq j \leq m_q, p'_i \sim_b q'_j$

We derive from these remarks,

$$\begin{aligned} before(\rho) &= \{p_0, \dots, p_{n_p}\} \cong_{\sim_b} before(\sigma) = \{q_0, \dots, q_{n_q}\} \text{ from 1 and 2} \\ after(\rho) &= \{p'_0, \dots, p'_{m_p}\} \cong_{\sim_b} after(\sigma) = \{q'_0, \dots, q'_{m_q}\} \text{ from 1 and 3} \end{aligned}$$

□

The following lemma is known as the *stuttering* lemma for branching bisimulation. We prove it for the rigid bisimulation.

Lemma 3.2 *Let ρ be a run such that $\Lambda(\rho) = \epsilon$ and $\alpha(\rho) \approxeq \beta(\rho)$. Then $\forall p \in states(\rho), \alpha(\rho) \approxeq p$.*

Proof: Let $p \in states(\rho)$. If $\alpha(\rho) \stackrel{a}{\Longrightarrow} q'$, for some $a \in A$, then $p \stackrel{\epsilon}{\Longrightarrow} \beta(\rho) \stackrel{a}{\Longrightarrow} p' \wedge p' \approxeq q'$, because $\alpha(\rho) \approxeq \beta(\rho)$. We have, for all $a \in A_\epsilon$, and for $f \in \{Post, Pre\}$:

$$\begin{aligned} f(p,a) &\sqsubseteq f(\alpha(\rho),a), \text{ since } \alpha(\rho) \stackrel{\epsilon}{\Longrightarrow} p \\ f(\beta(\rho),a) &\sqsubseteq f(p,a), \text{ since } p \stackrel{\epsilon}{\Longrightarrow} \beta(\rho) \\ f(\alpha(\rho),a) &\equiv f(\beta(\rho),a), \text{ since } \alpha(\rho) \approxeq \beta(\rho) \end{aligned}$$

So $f(p,a) \equiv f(\alpha(\rho),a)$. Thus, $\alpha(\rho) \approxeq p$. □

Another useful lemma is the so-called *X-lemma* enjoyed by branching bisimulation, when expressed as a weak back and forth bisimulation over runs [5], that we prove in our settings.

Lemma 3.3 *Let $p \approxeq q$, $a \in A_\epsilon$, $\rho \in obs(p,a)$ and $\sigma \in obs(q,a)$ such that $before(\rho) \cong before(\sigma)$. Let $p' \in before(\rho), q' \in before(\sigma)$ such that $p' \approxeq \beta(\sigma), q' \approxeq \beta(\rho)$. Then $p' \approxeq q'$ and $\beta(\rho) \approxeq \beta(\sigma)$.*

Proof: We just have to prove that $p' \approxeq q'$. $\beta(\rho) \approxeq \beta(\sigma)$ comes then from the fact that $\approxeq$ is an equivalence relation.
For all $a \in A$, if $p' \stackrel{a}{\Longrightarrow} p''$ then as $p' \approxeq \beta(\sigma)$, there exists $q'', q' \stackrel{\epsilon}{\Longrightarrow} \beta(\sigma) \stackrel{a}{\Longrightarrow} q'' \wedge p'' \approxeq q''$. The symmetric case holds in the same way. Now we have by the hypothesis, where $a \in A_\epsilon$ and $f \in \{Pre, Post\}$:

$$\begin{aligned} f(\beta(\rho),a) &\sqsubseteq f(p',a), \text{ since } p' \stackrel{\epsilon}{\Longrightarrow} \beta(\rho) && (3.1) \\ f(q',a) &\equiv f(\beta(\rho),a), \text{ since } q' \approxeq \beta(\rho) && (3.2) \\ f(\beta(\sigma),a) &\sqsubseteq f(q',a), \text{ since } q' \stackrel{\epsilon}{\Longrightarrow} \beta(\sigma) && (3.3) \\ f(p',a) &\equiv f(\beta(\sigma),a), \text{ since } p' \approxeq \beta(\sigma) && (3.4) \end{aligned}$$

From which we derive:

$$\begin{aligned} f(q',a) &\sqsubseteq f(p',a), \text{ from (1) and (2)} \\ f(p',a) &\sqsubseteq f(q',a), \text{ from (3) and (4)} \end{aligned}$$

That is $f(p',a) \equiv f(q',a)$, thus $p' \approxeq q'$ □

Lemma 3.4 *If $p \approxeq q$ then $\forall \rho \in obs(p,a), a \in A_\epsilon, \exists \sigma \in obs(q,a)$ such that:*

$$(\beta(\rho) \approxeq \beta(\sigma)) \wedge (before(\rho) \cong before(\sigma)) \wedge (after(\rho) \cong after(\sigma))$$

Proof: By construction. Given a run $\rho \in obs(p,a)$ we construct a run $\sigma \in obs(q,a)$ verifying the conditions above in the following way: First, as $p \approxeq q$, we have $Post(p,a) \equiv Post(q,a)$. So if we call $\rho_0 = left(\rho)$, we know that there exists $\sigma_0 = left(\sigma')$ for some $\sigma' \in obs(q,a)$ such that $states(\rho_0) \cong states(\sigma_0)$. By the X-lemma 3.3, $\beta(\rho_0) \approxeq \beta(\sigma_0)$. As $\beta(\rho_0) \xrightarrow{a} \alpha(right(\rho))$, we have that $\beta(\sigma_0) \xrightarrow{a} q'$ and $\alpha(right(\rho)) \approxeq q'$. Let $\sigma_1 = \sigma_0(\beta(\sigma_0),(\beta(\sigma_0),a,q'))$. As we have $\alpha(right(\rho)) \approxeq \beta(\sigma_1)$, then there exists $\sigma_2 \in obs(\beta(\sigma_1),\epsilon)$ such that $states(right(\rho)) \cong states(\sigma_2)$. We just let the run $\sigma = \sigma_1\sigma_2$. □

We state now the equality between $\approxeq$ and $\sim_b$.

Theorem 3.1

$$p \approxeq q \iff p \sim_b q$$

Proof:
$p \approxeq q \Leftarrow p \sim_b q$:

Since any branching bisimulation is also a weak bisimulation, we have $\forall a \in A_\epsilon$:

$$(p \overset{a}{\Longrightarrow} p') \Rightarrow (\exists q', q \overset{a}{\Longrightarrow} q' \wedge p' \sim_b q')(\textit{and vice-versa})$$

This covers the conditions (2) of definition 3.7. Lemma 3.1 covers the condition (1) of that definition.

$p \approxeq q \Rightarrow p \sim_b q$:

Let's see what happen whenever $p \overset{a}{\longrightarrow} p'$, for some p'.

1. Either $a = \tau$ and $p' \approxeq q$: then the branching bisimulation condition is fulfilled.
2. Or, let $\rho = (p, (p, a, p'))$. By lemma 3.4, $\exists \sigma \in obs(q, a), (\beta(\rho) \approxeq \beta(\sigma)) \wedge (f(\rho) \cong f(\sigma)), f \in \{before, after\}$. Let $\sigma = (q, (q \overset{a}{\Longrightarrow} q'))$. Two cases:
   (a) $a = \tau$: in this case, $before(\rho) = \{p, p'\}$ and let $before(\sigma) = \{q = q_0, \ldots, q_n = q'\}$. We have then $\forall i, 0 \leq i \leq n, p \approxeq q_i \vee p' \approxeq q_i$. By the stuttering lemma 3.2, there exists $k, l, 0 \leq k < l \leq n, \forall i, 0 \leq i \leq k, p \approxeq q_k, \forall l, k < l \leq n, p' \approxeq q_l$. This proves branching bisimulation of p and q in this case.
   (b) $a \neq \tau$: in this case, $before(\rho) = \{p\}$. We have then $\forall r \in before(\sigma), p \approxeq r$. Also, as $after(\rho) = \{p'\}$, we derive $\forall r \in after(\sigma), p \approxeq r$. This completely proves the branching bisimulation conditions for this case.

$\square$

4. An Algorithm for Rigid Bisimulation

We are now interested in providing an algorithm to decide $\approxeq$. Its core will be the algorithm for deciding strong bisimulation. The goal is to work only on the rigid lts obtained from the two lts that have to be compared. Without loss of generality, we work only with lts that do not contain τ-cycles.

4.1 The Algorithm

The Problem
Starting from a partition Π_0 of Q, the problem is stated as follow:
Where $\sim_\Pi$ is the equivalence induced by a partition Π, find the coarsest partition Π_f satisfying:

1. Π_f refines Π_0
2. if $p \sim_{\Pi_f} q$ then
   (a) $\forall a \in A_\epsilon, Pre(p, a) \equiv_{\Pi_f} Pre(q, a) \wedge Post(p, a) \equiv_{\Pi_f} Post(q, a)$
   (b) $\forall a \in A, p \overset{a}{\Longrightarrow} p' \Rightarrow (\exists q', q \overset{a}{\Longrightarrow} q' \wedge p' \sim_{\Pi_f} q')$ (*and vice-versa*)

For this problem, we propose an algorithm essentially based on the algorithm used to decide strong bisimulation that works on a decorated rigid lts.

The General Algorithm
This algorithm starts with an initial partition Π_0 of Q and refines it until a stable partition Π_f is found. Its scheme is displayed on figure 0.2 (where Π_0 is initialized as the universal relation, but it can be any partition of Q).

The Stability Notion
Here we make it more precise the stability notions used in the algorithm above.

```
$\Pi_0$ := $Q \times Q$
$\Pi$ := $\Pi_0$
while $\Pi$ not stable do {
   Find $B$ not stable
   $\Pi$ := $Ref(\Pi, B)$
}
```

Figure 0.2: Algorithm to compute the stable partition

Definition 4.1 *Let $\Pi = \cup_{i \in [1..n]} B_i$ be a partition on $Q \times Q$. We say that Π is B-stable if and only if*

1. $\forall a \in A_\epsilon, \forall B' \in \Pi, in_a(B') \cap B = \emptyset \vee in_a(B') \cap B = B$

2. $\forall a \in A_\epsilon, \forall q_1, q_2 \in B, Pre_1(q_1, a) \equiv_\Pi Pre_2(q_2, a) \wedge Post_1(q_1, a) \equiv_\Pi Post_2(q_2, a)$

Π is stable exactly when it is B-stable for each of its block B.

```
$\Pi_0$ := $Q \times Q$
$\Pi$ := $\Pi_0$
do {
   $\Pi'$ := RCP($\Pi$)
   $\Pi$ := CPP($\Pi'$)
} while ($\Pi \neq \Pi'$)
```

Figure 0.3: The two steps algorithm to compute the stable partition

The two levels stability notions leads to an algorithm where finding unstable blocks is achieved in two steps:

- the first step consists of the classical search of unstable blocks by means of state behaviors, like in the classical Relational Coarset Partition (RCP) algorithm. Indeed, the RCP algorithm is applied until stability is reached.
- the second steps looks for blocks unstable with respect to the *Pre/Post* sets. If no such a block is found the algorithm terminates. Otherwise, the partition is refined with respect to these unstable blocks and the two steps are re-applied starting with the RCP. We call the second step CPP (*Coarsest Pre/Post Partition*).

The two steps algorithm is displayed in figure 0.3.

The Refinement Steps

The first step of the algorithm applies the RCP refinement. It checks stability with respect to condition 1 of definition 4.1. When the partition Π is not stable w.r.t a block B for an action a, then $\exists B'$ such that $\emptyset \neq B_1 = (in_a(B') \cap B) \neq B$. In this case, refining Π is replacing B in Π blocks B_1 and B_2. The second step checks the stability condition 2 of definition 4.1. When the partition Π is not stable w.r.t a block B for an action a in this sense, B is then split into as many blocks as needed to gather states that have the same *Pre/Post* sets for the action a.

Now we have to settle the question of the efficiency of the CPP refinement step, that for the RCP one being well known. The second step performs the comparison of the *Pre/Post* sets between the states that are in the same equivalence class after the first step. The naive approach would be that of comparing these sets one by one manner; but this would obviously be too costly. However, by noticing some properties of the configuration of the *Pre/Post* sets after the first step it is possible to devise efficient strategies to efficiently compare these sets.

Comparing the *Pre/Post* sets

We state some properties about the configuration of the *Pre/Post* sets under weak bisimulation.

Lemma 4.1 *If $p \approx q$ and $f \in \{Pre, Post\}$, then*

$$\forall a \in A_\epsilon, \forall \Theta_1 \in f(p,a), \exists \Theta_2 \in f(q,a), \Theta_1 \leq_\approx \Theta_2$$

Proof: Suppose $\exists \Theta_1 \in Post(p,a), \forall \Theta_2 \in Post(q,a), \Theta_1 \not\leq_\approx \Theta_2$. In another words, $\exists \rho \in obs(p,a), \forall \sigma \in obs(q,a), after(\rho) \not\approx after(\sigma)$. This clearly contradicts $p \approx q$. The same reasoning on *Pre* sets leads to the same contradiction. □

Proposition 4.1 *If $p \approx q$ and $f \in \{Pre, Post\}$, then*

$$\forall a \in A_\epsilon, \forall \Theta_1 \in f(p,a), \exists \Theta_1' \in f(p,a), \Theta_1 \leq_\approx \Theta_1', \exists \Theta_2 \in f(q,a), \Theta_1' \cong_\approx \Theta_2$$

Proof: Let $\Theta_1 \in f(p,a)$. By lemma 4.1, $\exists \Theta_2 \in f(q,a), \Theta_1 \leq_\approx \Theta_2$. As $\approx$ is symmetric, $\exists \Theta_1' \in f(p,a), \Theta_2 \leq_\approx \Theta_1'$. By repeating the reasoning, we exhibit a maximal chain (because the *Pre/Post* sets are finite) $\Theta_1 = \Theta_1^1, \Theta_2 = \Theta_2^1, \ldots, \Theta_1^k, \Theta_2^k$ such that

1. $\forall i, 1 \leq i \leq k, \Theta_1^i \in f(p,a), \Theta_2^i \in f(q,a)$,
2. $\forall i, 1 \leq i < k, \Theta_1^i \leq_\approx \Theta_2^i \leq_\approx \Theta_1^{i+1} \leq_\approx \Theta_2^{i+1}$.
3. $\forall \Theta \in f(p,a), \Theta_1^k \leq_\approx \Theta \Rightarrow \Theta = \Theta_1^k$,
4. $\forall \Theta \in f(p,a), \Theta_2^k \leq_\approx \Theta \Rightarrow \Theta = \Theta_2^k$

Still using lemma 4.1, we end at $\Theta_1^k \cong \Theta_2^k$. We just take $\Theta_1' = \Theta_1^k$. □

To compare the *Pre/Post* sets, we shall consider their projection into the equivalence classes of the states they contain, i.e.,$[Pre]_\approx, [Post]_\approx$. Let $C = \{C_1, \ldots, C_k\}$ the equivalence classes of the largest weak bisimulation partition. Let $<$ be an ordering on C, such that $C_i < C_j \iff C_i \overset{\epsilon}{\Longrightarrow} C_j$. In the case $C_i \overset{\epsilon}{\not\Longrightarrow} C_j \wedge C_j \overset{\epsilon}{\not\Longrightarrow} C_i$, we let $C_i < C_j \iff i < j$. It is easy to check that $<$ is a total ordering over C. We also considere the underlying lexicographic ordering over the strings of $C^\star$. We first sort each element of $[Pre]_\approx, [Post]_\approx$ to get increasing sequences of classes, and sort $[Pre]_\approx, [Post]_\approx$ with the lexicographic order. We write $\widehat{[f]}_\approx, f \in \{Pre, Post\}$ for the sorted set obtained from $[f]_\approx$. From proposition 4.1, a string $c \in \widehat{[f]}_\approx$ is either a suffix of another string $c' \in \widehat{[f]}_\approx$ or is said *maximal.* Still from the proposition, each of the *Pre/Post* sets of two weakly equivalent states have the same maximal strings. Stability is enjoyed when the states of a given block have exactly the same strings in their projected-sorted *Pre/Post* sets.

The basic idea here is that of finding the coarsest partition that corresponds to weak bisimilar states after the first step. Then, simple list processing and string comparison can be used to check efficiently the second stability step. The point is that we do not get actually the weak partition after the first step, but rather an "almost" weak partition since the refinement is operated on all actions but ϵ. This is due to our definition of the rigid bisimulation and to the fact that we ignore the ϵ-transitions in the conditions relative to state behaviors. However, ϵ is considered in the *Pre/Post* conditions and we saw how it covers the behavior condition over ϵ (see proposition 3.2).

The following proposition gives us a way for finding the weak partition after the first step by adding to RCP a test over the $\epsilon - Pre/Post$ sets.

Proposition 4.2 *$p \approx q$ if and only if*

1. $\forall a \in A, p \overset{a}{\Longrightarrow} p' \Rightarrow (\exists q', q \overset{a}{\Longrightarrow} q', q \approx q')$ (and vice versa)
2. $\overline{[Post(p,\epsilon)]_{\approx}} = \overline{[Post(q,\epsilon)]_{\approx}}$

Proof: Direct from lemma 4.1. □

Thus, in the RCP step, we add the treatment of ϵ by the *Post* sets equivalence classes comparison. The second steps deals with a weak partition and can operate efficiently.
As usual the above algorithm can be used to minimize a lts with respect to the computed equivalence. Indeed, after the computation of the equivalence partition, one can get the minimal equivalent lts to the original one by shrinking the equivalence classes into one state and project the transition relation over this one state per class selection. Also, it can be used to decide if two lts' are rigid bisimilar (or branching bisimilar) by just applying the algorithm on the disjoint union of the two lts' to compare: then the two systems are equivalents exactly when their respective initial states belong to the same equivalence class in the final partition.

4.2 Correctness

The correctness of the previous algorithm is given by the following theorems, where $\sim_{\Pi_f}$ is the equivalence relation induced by the final partition Π_f obtained from the previous algorithm.

Theorem 4.1 *The algorithm of figure 0.2 terminates after at most $n - |\Pi_0|$ refinement steps. It ends with the coarsest stable partition refining Π_0. We call the ending partition Π_f.*

We do not discuss termination of the algorithm since it relies on the same conditions of the standard algorithm to decide bisimulation. However, we show that $\sim_{\Pi_f}$ corresponds to $\approxeq$.

Theorem 4.2

$$p \approxeq q \iff p \sim_{\Pi_f} q$$

Proof:
The proof follows the definition of $\approxeq$ and the two stability condition exposed in definition 4.1.

□

4.3 Discussion on Complexity

Let m represent the number of transitions and n the number of states of the transition system over which the time and space complexity of algorithms are analyzed.
We know that the best algorithm to decide strong bisimulation on finite transition systems is due to Paige and Tarjan. Its complexity is $\mathcal{O}(m.\log n)$, where m and n are the number of transitions and states respectively. In the branching bisimulation case, the best complexity result is knows as $\mathcal{O}(m.n)$, with the algorithm of Groote and Vaandrager.
We want, however, make two remarks on complexity.

- First the complexity evaluation has to be made on rigid graphs, obtained from a τ-clean graph, after an extension of the transition relation and after ϵ moves removals. Thereafter, the number of state/transitions may have changed. This is one difficult point in the sense that it makes it hard to compare the complexity of our algorithm and those over the original graphs. The problem is similar to the one we have when comparing weak observational equivalence with other observational equivalences, for which we also compute a transitive closure after τ-cleaning.

- Nevertheless, we can apply Paige and Tarjan on our decorated-rigid structures. Then, it comes downs to evaluate the space complexity of this decorating information on states and the time complexity of their comparison.

The Time Complexity
The first step has time complexity $\mathcal{O}(m.\log n)$ if one uses Paige and Tarjan algorithm. The second step is decomposed as follow:

The projection This is the step replacing each state by the equivalence class it belongs to. This can be realized in $\mathcal{O}(m)$, by visiting all the *Pre/Post* sets.

The sorting In [12], an algorithm is described to perform lexicographic ordering having time complexty $\mathcal{O}(m' + k)$, where k is the number of equivalence classes, and $m' = n.\log_l n$, where l is the number of transition labels.

Thus, the global time complexity is $\mathcal{O}(m.\log n)$.
After the first step (RCP), if no block is split within the second step (CPP) then we have finished. This corresponds to the cases where weak bisimulation and branching bisimulation coincide. It has been pointed out in [9] that in practice this is the case for a large class of transition graph representing parallel and communicating processes behaviors.
The Space Complexity
This is the weak theoretical point of our algorithm; we deal with an extended transition relation, computed by a transitive closure of the τ-relation and composition of relations to get the weak transition relation. However, we are not completely in the same situation of weak bisimulation since we remove the τ-transitions, even though the ϵ relation is represented *via* the *Pre/Post* sets. Morover, we can avoid the computation of the whole weak transition relation by using techniques introduced in [2]. Doing so, analyzing the space complexity becomes complex. Putting in practice the algorithm should tell us more.

5. Conclusion and Future Work
An alternative characterization of branching bisimulation relying on how branching bisimulation relates two states and throws light on its differences with weak bisimulation. Another advantage of our alternative definition is its possible parameterization. If one fixes the classical bisimulation scheme then different equivalences can be obtained by varying the way *Pre/Post* sets are related. For obtaining branching bisimulation we had to ask that the *Pre/Post* sets be bisimilar too. Obviously milder alternative are possible.
The main result of the paper is an algorithm that compute efficiently branching bisimulation. It is based on strong bisimulation checking, and we achieve a time complexity that improves the best known one for by using the Paige and Tarjan techniques [12] adapted to strong bisimulation. This is not possible for the original algorithm. However, we must say that our solution has a weak point in its space complexity. We need additional spaces to support the extended transition relation and the *Pre/Post* sets representation.
As future work, we shall study the spectrum of equivalences covered by our equivalence when modifying the requirement on the node labels. Also, further work shall be done to improve of the space complexity of the algorithm. Particularly, we shall study how to minimize this information and remove redundant *Pre/Post* sets. We also plan an implementation using *Binary Decision Diagrams*, an efficient data structure used to represent sets implicitly through their characteristic formula. This shall be done by extending the *symbolic bisimulation algorithms* from [3].
Further work shall contain the implementation issues together with detailed complexity analysis. We shall then compare our algorithm with the classical algorithm for branching bisimulation over practical case studies.

REFERENCES

1. J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.
2. A. Bouali. Weak and branching bisimulation in fctool. Technical Report 1575, INRIA, 1991.
3. A. Bouali and R. de Simone. Symbolic bisimulation minimisation. In *Fourth Workshop on Computer-Aided Verification*, Montreal, 1992.
4. R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica,* 24:211–237, 1987.
5. R. De Nicola, U. Montanari, and F.W. Vaandrager. Back and forth bisimulations. In Baeten and Klop [1], pages 152–165.
6. R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [1], pages 278–297.
7. R.J. van Glabbeek. The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.
8. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.
9. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M. Paterson, editor, *Proceedings* 17th *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer-Verlag, July 1990.
10. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
11. R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.
12. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing,* 16(6):973–989, 1987.
13. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5th *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.