Formal Aspects of Computing (1992) 3: 1–000
© 1992 BCS

Testing Equivalence as a Bisimulation Equivalence

Rance Cleaveland

Computer Science Department
North Carolina State University
Raleigh, NC 27695-8206
USA

Matthew Hennessy

Computer Science Department
University of Sussex
Falmer, Brighton BN1 9QH
ENGLAND

Keywords: Process algebras; finite-state systems; behavioural preorders; automated verification

Abstract.
In this paper we show how the testing equivalences and preorders on transition systems may be interpreted as instances of generalized bisimulation equivalences and prebisimulation preorders. The characterization relies on defining transformations on the transition systems in such a way that the testing relations on the original systems correspond to (pre)bisimulation relations on the altered systems. On the basis of these results, it is possible to use algorithms for determining the (pre)bisimulation relations in the case of finite-state transition systems to compute the testing relations.

1. Introduction

A common approach to the verification of systems involves the use of "high-level" systems as specifications of lower-level ones. The higher-level system describes abstractly the behaviour required of the implementation, while the lower-level

Correspondence and offprint requests to: R. Cleaveland and M. Hennessy

one details the proposed implementation. To establish that the proposed implementation satisfies its specification it is then necessary to prove a relationship between the higher-level and lower-level processes. Many relationships have been proposed in the literature as being suitable. They are either equivalences, in which case one must show that the implementation provides *exactly* the behaviour stipulated by the specification, or preorders, in which case one shows that the implementation provides *at least* the behaviour required.

Within this relation-based framework the two most developed schools of thought are the following.

1. *Bisimulations* [Mil89, Par81, Wal88] Equivalences and preorders are given in terms of recursively defined relations called *bisimulations* and *prebisimulations*, respectively. The formulation is mathematically elegant and yields efficient algorithms for determining if finite-state processes are related [KaS90, PaT87]. However, these relations are often too discriminating; two processes may be deemed unrelated even though there is no practical way of ascertaining this to be the case. In addition there is no smooth connection between the preorders and the equivalences.
2. *Testing/Failures* [DeH84, Hen88, Hoa85] Equivalences and preorders are defined in terms of *tests* which processes may or must satisfy. This notion is intuitively appealing and has led to a well-developed mathematical theory of processes that ties together the equivalences and preorders. However, no characterization of these equivalences has led to an algorithmic solution for finite-state processes.

The object of this paper is to describe algorithms for checking the various testing preorders and equivalences for finite-state processes. We do so by describing a bisimulation/prebisimulation-based characterization of these relations that relies on the use of appropriate *process transformations*. Intuitively, the transformations eliminate nondeterminism from processes and label states in the new transition systems with certain information; to compute testing relations, then, one uses a general bisimulation or prebisimulation checker on the transformed systems. As a consequence of our approach any implementation of a bisimulation/prebisimulation checker (of which there are several) can be easily adapted to check for the testing relations; in fact, the technique has been used in the Concurrency Workbench [CPS89, CPS90, CPS92], a general-purpose tool for the verification of finite-state processes. Moreover, although it is well-known that the problem of testing equivalence is PSPACE-complete [KaS90], the theoretical complexity appears not to be problematic in practice; our experience with a variety of examples has been that, in addition to being semantically desirable, the testing relations are a computationally viable means of verifying processes.

The remainder of the paper is organized as follows. Section 2 describes the model of processes that we use and defines the various equivalences and preorders in terms of it. Section 3 defines the process transformations that we then use in Section 4 to characterize the testing relations in terms of bisimulations and prebisimulations. Section 5 sketches how these results may be used to automate the testing relations in the case of finite-state processes, while Section 6 presents an extended example illustrating these concepts (and, incidentally, shows two processes that are testing equivalent but not bisimulation equivalent). Section 7 presents our conclusions.

2. Processes, Equivalences and Preorders

Our first task is to define precisely the notion of process and the various process equivalences and preorders that we examine in this paper. These definitions all rely on *labeled transition systems.*

Definition 2.1. A labeled transition system is a triple $\langle P, Act, \rightarrow \rangle$, where

1. P is a set of (process) states,
2. Act is a set of actions containing a distinguished *silent* action, τ, and
3. $\rightarrow \subseteq P \times Act \times P$ is the *transition relation.*

Intuitively, P represents the set of possible computation states, Act contains the actions that computations may consist of, and $\rightarrow$ describes the state transitions that may result from the execution of an action in a state. The action τ represents an *internal* computation step. We shall write $p \stackrel{a}{\rightarrow} p'$ in lieu of $\langle p, a, p' \rangle \in \rightarrow$, and we shall on occasion use $p \stackrel{a}{\rightarrow}$ to mean that there is a p' with $p \stackrel{a}{\rightarrow} p'$ and $p \stackrel{a}{\not\rightarrow}$ if no such p' exists. For technical convenience we shall also only consider *finite-branching* transition systems in this paper; formally, this means that for any $p \in P$, $|\{ p' \mid \exists a.\ p \stackrel{a}{\rightarrow} p' \}| < \infty$.

Given $\langle P, Act, \rightarrow \rangle$, any $p \in P$ also defines a labeled transition system

$$\langle P_p, Act, \rightarrow_p \rangle,$$

where $P_p \subseteq P$ consists of the states reachable from p via some finite sequence of transitions and $\rightarrow_p$ is the restriction of $\rightarrow$ to P_p. In this way, p defines a *process* whose initial state is p and whose operational behaviour is described by $\rightarrow_p$. Accordingly, we shall often refer to elements of P as processes; moreover, if $| \rightarrow_p | < \infty$ then we shall say that p is *finite-state.* Note that if $| \rightarrow_p | < \infty$ then $|P_p| < \infty$.

2.1. Bisimulations and Prebisimulations

We now define families of semantic equivalences and preorders that relate processes (states in a transition systems) on the basis of their behaviour. In what follows, assume that $\langle P, Act, \rightarrow \rangle$ is a labeled transition system.

Bisimulation equivalence, $\sim$, is a behavioural equivalence on states that is defined in terms of relations called *bisimulations.* Intuitively, a bisimulation may be thought of as a *matching* between states that has the property that if two states are matched then each a-derivative of each state must be matched by some a-derivative of the other. Here we define a slight generalization of bisimulation; a Π-bisimulation imposes the additional constraint that two matched states must also satisfy the specified binary relation Π, which is meant to relate states on the basis of information that they may contain.

Definition 2.2. Let $\Pi \subseteq P \times P$ be a relation. Then R is a Π-bisimulation if $R \subseteq \Pi$ and pRq implies the following.

1. $p \stackrel{a}{\rightarrow} p' \Rightarrow \exists q'.\ q \stackrel{a}{\rightarrow} q' \wedge p' R q'$.
2. $q \stackrel{a}{\rightarrow} q' \Rightarrow \exists p'.\ p \stackrel{a}{\rightarrow} p' \wedge p' R q'$.

Notice that if $\Pi = P \times P$ then a Π-bisimulation is a bisimulation in the usual sense [Mil83, Mil89, Par81].

The relation $\sim_\Pi$ is defined in terms of Π-bisimulations in the same way that $\sim$ is defined in terms of bisimulations.

Definition 2.3. $p \sim_\Pi q$ if and only if there is a Π-bisimulation R with pRq.

The relation $\sim_\Pi$ is the largest Π-bisimulation; moreover, if Π is an equivalence relation, then so is $\sim_\Pi$. Again, if $\Pi = P \times P$ then $\sim_\Pi$ coincides with $\sim$, the usual notion of bisimulation equivalence.

The *prebisimulation preorder*, $\sqsubseteq_\approx$, relates processes in terms of relations called prebisimulations [Hen81, Wal88]. The definition of prebisimulation is similar to that of bisimulation; the chief difference is that the "matching conditions" a prebisimulation must satisfy are relaxed somewhat to allow for the possibility that the behaviour of a state may not be completely defined for certain actions. Formally, let $\Uparrow \subseteq P \times Act$ be a predicate called the *divergence predicate*; we shall write $p \Uparrow a$ in lieu of $\langle p, a\rangle \in \Uparrow$ and $p \Downarrow a$ in lieu of $\neg(p \Uparrow a)$. Intuitively, if $p \Uparrow a$ is true then the behavior of p on a is not completely defined, and thus a prebisimulation relating p to q need not match every a-transition of q with some a-transition in p. We in fact define a generalization of prebisimulation; in a $\langle \Pi, \Psi\rangle$-prebisimulation, two matched states must satisfy Π, and Ψ may be used to relax requirements on the transitions of p that q must match when p is matched to q.

Definition 2.4. Let $\Pi \subseteq P \times P$ and $\Psi \subseteq P \times Act$. A relation R is a $\langle \Pi, \Psi\rangle$-prebisimulation if $R \subseteq \Pi$ and pRq implies the following.

1. $\langle q, a\rangle \in \Psi \Rightarrow [\langle p, a\rangle \in \Psi \wedge (p \stackrel{a}{\rightarrow} p' \Rightarrow \exists q'.\ q \stackrel{a}{\rightarrow} q' \wedge p' R q')]$.
2. $p \Downarrow a \Rightarrow [q \Downarrow a \wedge (q \stackrel{a}{\rightarrow} q' \Rightarrow \exists p'.\ p \stackrel{a}{\rightarrow} p' \wedge p' R q')]$.

If $\Pi = P \times P$ and $\Psi = P \times Act$ then a $\langle \Pi, \Psi\rangle$-prebisimulation is a prebisimulation in the usual sense. We may now define $\sqsubseteq_{\langle \Pi, \Psi\rangle}$ as follows.

Definition 2.5. $p \sqsubseteq_{\langle \Pi, \Psi\rangle} q$ if and only if there is a $\langle \Pi, \Psi\rangle$-prebisimulation R with pRq.

Again, $\sqsubseteq_{\langle \Pi, \Psi\rangle}$ is the largest $\langle \Pi, \Psi\rangle$-prebisimulation. Furthermore, if Π is a preorder then so is $\sqsubseteq_{\langle \Pi, \Psi\rangle}$, for any Ψ.

Several preorders in the literature can be seen to be instances of the $\sqsubseteq_{\langle \Pi, \Psi\rangle}$ preorder. For example, by taking $\Pi = P \times P$ and $\Psi = P \times Act$ then $\sqsubseteq_{\langle \Pi, \Psi\rangle}$ coincides with $\sqsubseteq_\approx$, the prebisimulation preorder. As another example, the simulation preorder can be obtained by setting $\Downarrow = \emptyset$ (so every state is considered to be "underspecified" for each action a), $\Pi = P \times P$, and $\Psi = P \times Act$. Note that when $\Downarrow = \emptyset$, the effect is to eliminate the checking of condition 2. in the definition of $\langle \Pi, \Psi\rangle$-bisimulation. Finally, the inverse simulation preorder results by taking $\Downarrow = P \times Act$, $\Pi = P \times P$, and $\Psi = \emptyset$; in this case, setting $\Psi = \emptyset$ essentially eliminates the need to consider condition 1. in the definition.

2.2. The Testing Relations

The testing preorders and equivalences are defined on the basis of two preorders, $\sqsubseteq_{may}$ and $\sqsubseteq_{must}$ [DeH84, Hen88]. These have an operational characterization in terms of the responses of processes to tests [DeH84, Hen88]; in this paper, however, we use other, albeit equivalent, definitions.

2.2.1. The May Preorder

The may preorder, $\sqsubseteq_{may}$, relates states in terms of the sequences of visible actions that are possible from the state (with any intervening internal actions ignored). In what follows, we use ϵ to denote the empty string.

Definition 2.6.

1. Let $s \in (Act - \{\tau\})^*$ be a sequence of visible actions. Then $\stackrel{s}{\Rightarrow}$ is defined inductively on the structure of s as follows.
   (a) $\stackrel{\epsilon}{\Rightarrow} = \stackrel{\tau}{\rightarrow}^*$, where $\stackrel{\tau}{\rightarrow}^*$ is the transitive and reflexive closure of $\stackrel{\tau}{\rightarrow}$.
   (b) $\stackrel{as'}{\Rightarrow} = \stackrel{\epsilon}{\Rightarrow} \circ \stackrel{a}{\rightarrow} \circ \stackrel{s'}{\Rightarrow}$, where $\circ$ denotes relational composition.
2. $L(p) = \{ s \in (Act - \{\tau\})^* \mid \exists p'.\ p \stackrel{s}{\Rightarrow} p' \}$ is the *language* of p.

So $p \stackrel{\epsilon}{\Rightarrow} q$ if p can perform some number (possible 0) of internal computation steps and evolve to q, while $p \stackrel{s}{\Rightarrow} q$ (s nonempty) if p can perform the actions listed in s with any number of intervening τ actions allowed in between and end up as q. The set $L(p)$ then contains all the visible sequences s that p can exhibit.

The may preorder can now be defined as follows.

Definition 2.7. Let p and q be states. Then $p \sqsubseteq_{may} q$ if $L(p) \subseteq L(q)$.

2.2.2. The Must Preorder

For deterministic processes (i.e. those that are incapable of internal computation and never have more than one a-transition for any given a), the language usually conveys all behavioural information of interest. However, for nondeterministic processes this is not necessarily the case; the different states a process can reach after executing a given sequence of actions can exhibit quite different behaviour. Moreover, processes capable of τ-actions may sometimes *diverge*, i.e. engage in an infinite internal computation without allowing any visible actions to be performed. In contrast to the may preorder, then, the must preorder is designed to be sensitive to these aspects of the behaviour of nondeterministic processes. The formal definition of this relation relies on the following notions.

Definition 2.8. Let $p \in P$ be a state, $s, s' \in (Act - \{\tau\})^*$ be sequences of visible actions, and $a \in Act$.

1. $S(p) = \{ a \in Act \mid p \stackrel{a}{\rightarrow} \}$.
2. $D(p, a) = \{ p' \mid p \stackrel{a}{\rightarrow} p' \}$. We also abuse notation slightly by writing $D(p) = \bigcup_{a \in Act} D(p, a)$.
3. The *acceptance set* of p after s is defined as follows.
$$\mathcal{A}(p, s) = \{ S(p') \mid p \stackrel{s}{\Rightarrow} p' \wedge p' \stackrel{\tau}{\not\rightarrow} \}$$
4. The *convergence relation* $p \downarrow s$ is defined inductively as follows.
   (a) $p \downarrow \epsilon$ if there is no infinite sequence $\langle p_i \rangle_{i \geq 0}$ such that $p \stackrel{\tau}{\rightarrow} p_0$ and $p_i \stackrel{\tau}{\rightarrow} p_{i+1}$.
   (b) $p \downarrow as'$ if $p \downarrow \epsilon$ and $p \stackrel{a}{\Rightarrow} p'$ implies $p' \downarrow s'$.

The definitions of $S(p)$ and $D(p, a)$ are straightforward. Notice that our assumption about finite-branching ensures that $D(p)$ is always finite. The acceptance

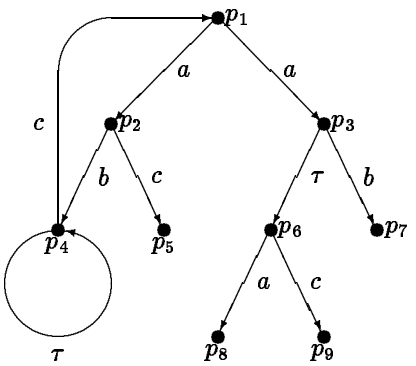


Fig. 1. A sample transition system.

set $\mathcal{A}(p,s)$ represents the set of "action capabilities" of p after s. Each set S in $\mathcal{A}(p,s)$ corresponds to a state p may reach by executing s and contains the set of next possible actions in that state. The fact that $\mathcal{A}(p,s)$ may contain more than one such set indicates that nondeterministic choices may occur during the execution of s; the more sets $\mathcal{A}(p,s)$ contains, the more nondeterministic p is in its execution of s. Note that if $s \notin L(p)$ then $\mathcal{A}(p,s) = \emptyset$. Finally, $p \downarrow s$ if it is impossible for p to perform an infinite sequence of τ actions as it attempts to "execute" s; it does *not* imply that $s \in L(p)$. The finite-branching assumption implies that when $p \downarrow s$, $\mathcal{A}(p,s)$ is a finite set of finite sets.

As an illustration of these definitions, consider the transition system given in Figure 1. We have that

$$\mathcal{A}(p_1, a) = \{\{b, c\}, \{a, c\}\},$$

since of the three states—p_2, p_3 and p_6—reachable from p_1 via $\stackrel{a}{\Rightarrow}$, only p_2 and p_6 are incapable of τ transitions and thus have their sets of actions included in the acceptance set. Moreover, $p_1 \downarrow ac$ holds, as does $p_1 \downarrow acc$ (even though $acc \notin L(p_1)$); on the other hand, $p_1 \downarrow ab$ does *not* hold, since p_4, which is one of the states reachable from p_1 via $\stackrel{ab}{\Rightarrow}$, is capable of an infinite τ computation.

The must preorder uses acceptance sets as the basis for relating processes on the basis of their relative nondeterminism. This is achieved using the following inclusion relation on sets of sets of actions in the [DeH84, Hen88]; intuitively, $A \subset\subset B$ if the A represents a "less nondeterministic" set of processes.

Definition 2.9. Let $A, B \subseteq 2^{Act}$. Then $A \subset\subset B$ if:

$$\forall S \in A.\, \exists S' \in B.\, S' \subseteq S.$$

So $\mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$ if for every stable state reachable from q via s there is some stable state reachable from p via s whose enabled actions are also enabled in the state associated with q. We now define $\sqsubseteq_{must}$ as follows.

Definition 2.10. $p \sqsubseteq_{must} q$ if $\forall s \in (Act - \{\tau\})^*.\ p \downarrow s \Rightarrow (q \downarrow s \wedge \mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s))$.

So $\sqsubseteq_{must}$ relates processes on the basis of their convergence behaviour, embodied in the predicate $\downarrow s$, and their relative degrees of nondeterminism, which is represented by the relationship between their acceptance sets. It is worth noting

that in the case of *strongly convergent* processes—namely, processes p such that for all s, $p\downarrow s$ — $p \sqsubseteq_{must} q$ implies that $q \sqsubseteq_{may} p$.

It also turns out that $\sqsubseteq_{must}$ can be defined with respect to "minimized" version of acceptance sets; we use this observation later when we discuss the computation of the testing preorders. Consider the operator *min* on acceptance sets defined by

$$min\ A = \{\, S \in A \mid \neg \exists S' \in A.\ S' \subset S \,\}.$$

So *min* A contains the elements of A having no proper subsets that are also elements of A. We have the following result.

Lemma 2.11. Let $A, B \subseteq 2^{Act}$.

1. $A \subset\subset B$ if and only if $min\ A \subset\subset min\ B$.
2. $A \subset\subset B$ and $B \subset\subset A$ if and only if $min\ A = min\ B$.

Proof. 1. ($\Rightarrow$) Suppose $A \subset\subset B$. Since $min\ A \subseteq A$ it follows that $\forall S \in min\ A.\ \exists S' \in B.\ S' \subseteq S$. Moreover, from the definition of *min* it follows that for every $S' \in B$ there is an $S'' \in min\ B$ with $S'' \subseteq S'$. These two facts imply that $\forall S \in min\ A.\ \exists S'' \in min\ B.\ S'' \subseteq S$, whence $min\ A \subset\subset min\ B$.
($\Leftarrow$) Now suppose that $min\ A \subset\subset min\ B$. Since $min\ B \subseteq B$ it follows that $\forall S \in min\ A.\ \exists S' \in B.\ S' \subseteq S$. $A \subset\subset B$ follows from this and from the fact that for any $S \in A$ there is an $S' \in min\ A$ with $S' \subseteq S$.

2. ($\Rightarrow$) Suppose $A \subset\subset B$ and $B \subset\subset A$. It is sufficient to show that $min\ A \subseteq min\ B$ and $min\ B \subseteq min\ A$. We show the former; the latter follows by symmetry. So let $S \in min\ A$. Since $A \subset\subset B$ it follows from (1.) that $min\ A \subset\subset min\ B$, and this means that there is an $S' \in min\ B$ with $S' \subseteq S$. Moreover, as $B \subset\subset A$, we may conclude that $min\ B \subset\subset min\ A$, and this implies that there is an $S'' \in min\ A$ with $S'' \subseteq S'$. As it is the case that $S'' \subseteq S' \subseteq S$ and both S'' and S are in $min\ A$, it follows that $S'' = S$, and hence $S' = S$, whence $S \in min\ B$.
($\Leftarrow$) Follows from the fact that $min\ A = min\ B$ implies that $min\ A \subset\subset min\ B$ and $min\ B \subset\subset min\ A$.
□

2.2.3. Other Testing Relations

Given the definitions of $\sqsubseteq_{may}$ and $\sqsubseteq_{must}$, the other testing relations, [Hen88], are defined as follows.

Definition 2.12. The testing preorder, $\sqsubseteq_t$, and the testing equivalences, $=_{may}$, $=_{must}$ and $=_t$, are defined by:

1. $p \sqsubseteq_t q$ if $p \sqsubseteq_{may} q$ and $p \sqsubseteq_{must} q$, and
2. $p =_i q$ if $p \sqsubseteq_i q$ and $q \sqsubseteq_i p$, for $i \in \{may, must, t\}$.

3. Acceptance Graphs

The bisimulation/prebisimulation-based accounts of the testing relations rely on transforming the transition systems in question in a suitable way. In this section

we describe the kinds of transition systems to be generated and transformations used to generate them. We first define the notion of a *deterministic* transition system.

Definition 3.1. Let $\langle P, Act, \rightarrow \rangle$ be a labeled transition system. Then $\rightarrow$ is *deterministic* if $\stackrel{\tau}{\rightarrow} = \emptyset$ and $|D(p,a)| \leq 1$ for each $p \in P$ and $a \in Act$.

If $\rightarrow$ is deterministic then we shall sometimes say that the corresponding labeled transition system is also deterministic.

Acceptance graphs, or *Agraphs* for short, are a particular class of deterministic labeled transition systems. In essence, each state in an Agraph is labeled with two pieces of information: a minimized acceptance set, and a boolean indicating whether the state is "convergent" (or *closed*, in the terminology of [Hen85]). The formal definition is the following.

Definition 3.2. A labeled transition system $\langle T, Act, \rightarrow \rangle$ is an Agraph if $\rightarrow$ is deterministic and the following hold for each $t \in T$.

1. t is labeled by two pieces of information, $t.acc$ and $t.closed$, with $t.acc \subseteq 2^{Act}$ and $t.closed$ a boolean.
2. $t.acc$ is finite, and each $S \in t.acc$ is finite.
3. $t.acc = min\, t.acc$.
4. $t.closed = true \iff t.acc \neq \emptyset$.
5. $t.closed = false \Rightarrow \forall t' \in D(t).\ t'.closed = false$.

The definition of Agraph should be compared with notion of acceptance tree in [Hen85, Hen88]. Notice that condition 5. ensures that if $t.closed$ holds, then $u.closed$ must hold of each predecessor u of t along $\rightarrow$. On occasion, we shall say that an Agraph node t is *closed* if $t.closed$ is true, and *open* otherwise.

Two variants of Agraphs are also of interest.

Definition 3.3.

1. Agraph $\langle T, Act, \rightarrow \rangle$ is a *strong* Agraph, or SAgraph, if for each $t \in T$, $t.closed = false \Rightarrow D(t) = \emptyset$.
2. Agraph $\langle T, Act, \rightarrow \rangle$ is a WAgraph if for each $t \in T$, $t.closed = false$.

In an SAgraph, no open node has any transitions, while a WAgraph contains no closed nodes at all.

3.1. Building Acceptance graphs

We now define a construction for generating Agraphs from arbitrary labeled transition systems; the procedure turns out to preserve certain language, acceptance set and convergence characteristics of the original system. We begin by defining the standard automata-theoretic notion of *ϵ-closure* of a set of states Q as follows.

$$Q^{\epsilon} = \{\, p \mid \exists q \in Q.\ q \stackrel{\epsilon}{\Rightarrow} p \,\}$$

Notice that $Q \subseteq Q^{\epsilon}$ for any Q, since for any p, $p \stackrel{\epsilon}{\Rightarrow} p$. We also extend the definitions of D and $\downarrow s$ to sets of states by saying

$$D(Q,a) = \bigcup_{q \in Q} D(q,a)$$

and using the notation

$$Q\downarrow \iff |Q| < \infty \wedge \bigwedge_{q\in Q} q\downarrow\epsilon$$

As before, we shall write $Q\uparrow$ if $\neg(Q\downarrow)$. Note that for $Q\downarrow$ to hold, Q must be finite. Now let $\mathcal{L} = \langle P, Act, \rightarrow\rangle$ be a labeled transition system. The Agraph construction that we are about to describe is very similar to the usual method from automata theory for building a deterministic automaton that is language-equivalent to a given nondeterministic one. Here, states in the Agraph will correspond to pairs consisting of booleans and sets of states in the original transition system. The boolean is used to represent divergence information. In addition each Agraph state is labeled with a minimized acceptance set. The formal definition of Agraph $\mathcal{T}(\mathcal{L}) = \langle T, Act, \rightarrow_T\rangle$ is as follows.

1. $T = \{\, \langle Q, b\rangle \in 2^P \times \mathit{boolean} \mid Q = Q^\epsilon \wedge (b = \mathit{true} \Rightarrow Q\downarrow) \,\}$.
2. For $t = \langle Q, b\rangle \in T$, define $t.\mathit{closed} = b$ and

$$t.acc = \begin{cases} \emptyset & \text{if } t.\mathit{closed} = \mathit{false} \\ \mathit{min}\,\{\, S(q) \mid q \in Q \wedge q \stackrel{\tau}{\not\rightarrow} \,\} & \text{otherwise} \end{cases}$$

3. For $t_1 = \langle Q_1, b_1\rangle$ and $t_2 = \langle Q_2, b_2\rangle$, $t_1 \stackrel{a}{\rightarrow_T} t_2$ exactly when the following hold.
   (a) $a \neq \tau$.
   (b) $(D(Q_1, a))^\epsilon = Q_2$.
   (c) $[b_1 = \mathit{false} \Rightarrow b_2 = \mathit{false}] \wedge [(b_1 = \mathit{true} \wedge b_2 = \mathit{false}) \Rightarrow Q_2\uparrow]$.

It is straightforward to verify that $\mathcal{T}(\mathcal{L})$ is an Agraph. The definition of $\downarrow$ ensures that $t.acc$ satisfies the finiteness properties required of Agraphs for each $t \in T$, while the construction of $\rightarrow_T$ ensures that it is deterministic.

As examples of this construction, consider the transition system given in Figure 1. We would have the following transitions in the associated Agraph.

$$\langle\{p_1\}, \mathit{true}\rangle \stackrel{a}{\rightarrow_T} \langle\{p_2, p_3, p_6\}, \mathit{true}\rangle \quad (1)$$

$$\langle\{p_2, p_3, p_6\}, \mathit{true}\rangle \stackrel{b}{\rightarrow_T} \langle\{p_4, p_7\}, \mathit{false}\rangle \quad (2)$$

$$\langle\{p_4, p_7\}, \mathit{false}\rangle \stackrel{a}{\rightarrow_T} \langle\{p_1\}, \mathit{false}\rangle \quad (3)$$

Notice that in transition 2, the boolean associated with $\{p_4, p_7\}$ is *false* because $p_4\uparrow\epsilon$. Also, in transition 3 the boolean associated with $\{p_1\}$ must be *false* because of the *false* associated with $\{p_4, p_7\}$.

It is possible to alter this construction slightly to generate $\mathcal{ST}(\mathcal{L})$ and $\mathcal{D}(\mathcal{L})$, which are an SAgraph and WAgraph, respectively. The alterations are the following.

- $\mathcal{ST}(\mathcal{L})$ Replace (3.c) with $b_1 = \mathit{true} \wedge (Q_2\downarrow \Rightarrow b_2 = \mathit{true})$.
- $\mathcal{D}(\mathcal{L})$ Replace (1.) with $T = \{\, \langle Q, \mathit{false}\rangle \mid Q = Q^\epsilon \,\}$.

The net effect of the $\mathcal{ST}$ alteration is to prevent the computation of transitions for open states. The $\mathcal{D}$ construction ensures that no state in $\mathcal{D}(\mathcal{L})$ will be closed.

3.2. Properties of Acceptance Graphs

In order to establish the properties enjoyed by $\mathcal{T}(\mathcal{L})$, $\mathcal{ST}(\mathcal{L})$ and $\mathcal{D}(\mathcal{L})$ with respect to $\mathcal{L}$, it is useful to make some definitions. Let $p \in P$.

$$\begin{aligned} t(p) &= \langle \{p\}^\epsilon, p \downarrow \epsilon \rangle \\ d(p) &= \langle \{p\}^\epsilon, false \epsilon \rangle \end{aligned}$$

$t(p)$ is meant to define the state in $\mathcal{T}(\mathcal{L})$ (and $\mathcal{ST}(\mathcal{L})$) corresponding to p, while $d(p)$ is meant to define the state in $\mathcal{D}(\mathcal{L})$ corresponding to p.

In deterministic graphs such as Agraphs, a sequence $s \in L(t)$ for a state t defines a unique state t'. Accordingly, we shall abuse notation and say that, for $s \in L(t)$, $D(t,s)$ is this state. We also redefine the predicate $\downarrow s$ for Agraphs in the following way.

$$\begin{aligned} t \downarrow \epsilon &\iff t.closed = true \\ t \downarrow as' &\iff t.closed = true \wedge (t \xrightarrow{a} t' \Rightarrow t' \downarrow s') \end{aligned}$$

Let $\mathcal{L} = \langle P, Act, \rightarrow \rangle$. The first lemma we state establishes a "language equivalence" property between $p \in P$ and its corresponding nodes in $\mathcal{T}(\mathcal{L})$ and $\mathcal{D}(\mathcal{L})$.

Lemma 3.4. Let $p \in P$.

1. Let $t(p)$ be the state associated with p in $\mathcal{T}(\mathcal{L})$. Then $L(p) = L(t(p))$.
2. Let $d(p)$ be the state associated with p in $\mathcal{D}(\mathcal{L})$. Then $L(p) = L(d(p))$.

Proof. By induction on the length of $s \in (Act - \{\tau\})^*$. □

In general, this property does not hold between p and its corresponding state in $\mathcal{ST}(\mathcal{L})$, because the fact that open nodes in SAgraphs have no outgoing edges means that some sequences can be "lost". However, we do have the following.

Lemma 3.5. Let $s \in (Act - \{\tau\})^*$.

1. Assume that $p \downarrow s$. Then $s \in L(p)$ if and only if $s \in L(t(p))$, where $t(p)$ is the state associated with p in $\mathcal{ST}(\mathcal{L})$.
2. $p \downarrow s$ if and only if $t(p) \downarrow s$, where $t(p)$ is the state associated with p in either $\mathcal{T}(\mathcal{L})$ or $\mathcal{ST}(\mathcal{L})$.
3. If $p \downarrow s$ and $s \in L(p)$ then $D(t(p), s).acc = min\ \mathcal{A}(p, s)$, where $t(p)$ is the state associated with p in either $\mathcal{T}(\mathcal{L})$ or $\mathcal{ST}(\mathcal{L})$.

Proof. We prove 1.; the others are left for the reader. We do so by establishing the following slightly stronger result.

Assume that $p \downarrow s$. Then $t(p) \downarrow s$, and $s \in L(p)$ if and only if $s \in L(t(p))$.

The proof proceeds by induction on s. Since $\epsilon \in L(q)$ for any q, the base case amounts to showing that $p \downarrow \epsilon$ if and only if $t(p) \downarrow \epsilon$, which follows directly from the definition of $t(p)$. For the inductive step, let $s = s'a$; the induction hypothesis states that if $p \downarrow s'$, then $t(p) \downarrow s'$ and $s' \in L(p)$ if and only if $s' \in L(t(p))$. Now assume that $p \downarrow s$; we must show that $t(p) \downarrow s$ and that $s \in L(p)$ if and only if $s \in L(t(p))$. It follows that $p \downarrow s'$, and we may apply the induction hypothesis to conclude that $t(p) \downarrow s'$ and that $s' \in L(p)$ if and only if $s' \in L(t(p))$. There are two cases to consider.

- $s \notin L(p)$. Then $s \notin L(t(p))$, since lemma 3.4(1.) guarantees that $s \notin L(t(p))$

when $L(t(p))$ is interpreted with respect to $\mathcal{T}(\mathcal{L})$ and for any node t, the language of t in $\mathcal{ST}(\mathcal{L})$ is contained in the language of t in $\mathcal{T}(\mathcal{L})$. We now must establish that $t(p)\downarrow s$. This, however, follows from the definition of $\downarrow s$, the fact that $t(p)\downarrow s'$, and the fact that $s \notin L(t(p))$.

- $s \in L(p)$. This implies that $s' \in L(p)$, whence we may conclude that $s' \in L(t(p))$ on the basis of the induction hypothesis. To show that $s \in L(t(p))$, then, it is sufficient to show that $D(t(p), s)$ has an a-transition emanating from it. Since $s' \in L(p)$ and $s \in L(p)$, it follows that the sets $S = \{\, p' \mid p \stackrel{s}{\Rightarrow} p' \,\}$ and $S' = \{\, p' \mid p \stackrel{s'}{\Rightarrow} p' \,\}$ are nonempty, and a simple inductive argument establishes that $D(t(p), s') = \langle S', \mathit{true}\rangle$ (note that $S' = S'^{\epsilon}$). It is also the case that $D(S', a)^{\epsilon} = S$ and that for no $p \in S$ is it the case that $p\uparrow$, and it therefore follows from the construction of $\mathcal{ST}(\mathcal{L})$ that $D(t(p), s') \stackrel{a}{\rightarrow}_{ST} \langle S, \mathit{true}\rangle$. Thus $s \in t(p)$ and $t(p)\downarrow s$.

□

4. Testing via Bisimulations and Prebisimulations

In this section we prove our main results linking the testing relations on labeled transition systems with appropriate Π-bisimulations and $\langle \Pi, \Psi\rangle$-prebisimulations on Agraphs, SAgraphs and WAgraphs. Let $\mathcal{L} = \langle P, Act, \rightarrow\rangle$ be an arbitrary labeled transition system. We shall use the following notation.

$$\begin{aligned} \mathcal{T}(\mathcal{L}) &= \langle T, Act, \rightarrow_T\rangle \\ \mathcal{ST}(\mathcal{L}) &= \langle T, Act, \rightarrow_{ST}\rangle \\ \mathcal{D}(\mathcal{L}) &= \langle D, Act, \rightarrow_D\rangle \end{aligned}$$

We now turn our attention to the preorders. To start with, we need to define the convergence predicate $\Downarrow a$ to be used in the definition of prebisimulation on Agraphs. We use the following.

$$t \Downarrow a \iff t.\mathit{closed} = \mathit{true}$$

Since this definition does not depend on a we shall abuse notation and write $t \Downarrow$.

Theorem 4.1. Define $\Pi = \{\, \langle t, u\rangle \mid t.\mathit{acc} = \emptyset \vee u.\mathit{acc} \subset\subset t.\mathit{acc} \,\}$, and let $p, q \in P$. Then the following hold.

1. $p \sqsubseteq\!\!\!\approx_{\mathit{must}} q$ if and only if $t(p) \sqsubseteq\!\!\!\approx_{(\Pi,\emptyset)} t(q)$ in $\mathcal{ST}(\mathcal{L})$.
2. $p \sqsubseteq\!\!\!\approx_{t} q$ if and only if $t(p) \sqsubseteq\!\!\!\approx_{(\Pi, T\times Act)} t(q)$ in $\mathcal{T}(\mathcal{L})$.

Proof. We shall only prove (1.); the proof of (2.) is similar and left to the reader.

($\Rightarrow$) Suppose that $p \sqsubseteq\!\!\!\approx_{\mathit{must}} q$; we must exhibit a $\langle \Pi, \emptyset\rangle$-prebisimulation R such that $t(p)\ R\ t(q)$. Define R as follows.

$$\begin{aligned} R \;=\; \{\, \langle t, u\rangle \mid \forall s \in (Act - \{\tau\})^*.\ t\downarrow s \Rightarrow \\ [u\downarrow s \wedge (s \in L(u) \Rightarrow s \in L(t) \wedge D(u,s).\mathit{acc} \subset\subset D(t,s).\mathit{acc})] \,\} \end{aligned}$$

We must first show that $t(p)\ R\ t(q)$. Fix $s \in (Act - \{\tau\})^*$ and assume that $t(p)\downarrow s$; by Lemma 3.5(2.) it must be the case that $p\downarrow s$ and, since $p \sqsubseteq\!\!\!\approx_{\mathit{must}} q$, $q\downarrow s$. Lemma 3.5(2.) guarantees that $t(q)\downarrow s$. Now assume in addition that $s \in L(t(q))$.

By Lemma 3.5(1.), $s \in L(q)$, and from this, the fact that $q \downarrow s$, and the definition of $\mathcal{A}(q,s)$, it follows that $\mathcal{A}(q,s) \neq \emptyset$. Since $p \sqsubseteq_{must} q$, $\mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$, and it must therefore be the case that $\mathcal{A}(p,s) \neq \emptyset$. These facts imply that $s \in L(p)$ and, by Lemma 3.5(2.), $s \in L(t(p))$. Moreover, since $D(t(p),s).acc = min\ \mathcal{A}(p,s)$ and $D(t(q),s).acc = min\ \mathcal{A}(q,s)$ (Lemma 3.5(3.)), Lemma 2.11(2.) allows us to conclude that $D(t(p),s).acc \subset\subset D(t(q),s).acc$. Thus $t(p)\ R\ t(q)$.

We now must establish that R is a $\langle \Pi, \emptyset \rangle$-prebisimulation. Assume that $t\ R\ u$; we must show the following.

1. $\langle t, u \rangle \in \Pi$, and
2. $t\Downarrow \Rightarrow [u\Downarrow \wedge (u \stackrel{a}{\rightarrow}_{ST} u' \Rightarrow \exists t'.\ t \stackrel{a}{\rightarrow}_{ST} t' \wedge t'\ R\ u')]$.

To show (1.), assume that $t.acc \neq \emptyset$. From the definition of Agraph it follows that $t.closed = true$, and hence $t \downarrow \epsilon$. By the definition of R, then, $u \downarrow \epsilon$. Moreover, as $\epsilon \in L(u)$ (trivially), $D(u,\epsilon).acc = u.acc \subset\subset t.acc = D(t,\epsilon).acc$. Therefore, $\langle t, u \rangle \in \Pi$.

To prove (2.), assume that $t\Downarrow$, i.e. $t \downarrow \epsilon$. From the definition of R it follows that $u \downarrow \epsilon$, and therefore $u\Downarrow$. So assume further that $u \stackrel{a}{\rightarrow}_{ST} u'$, i.e. $D(u,a) = u'$; we must show that $D(t,a) = t'$ exists and that $t'\ R\ u'$. If t' does not exist then we get a contradiction, since in this case $t \downarrow a$ (since $t \downarrow \epsilon$), $a \in L(u)$, and $a \notin L(t)$. So t' exists; we now must show that $t'\ R\ u'$. Fix $s \in (Act - \{\tau\})^*$, and assume that $t' \downarrow s$. Since $\rightarrow_{ST}$ is deterministic, it follows that $t \downarrow as$, and as $t\ R\ u$ this means that $u \downarrow as \wedge (as \in L(u) \Rightarrow as \in L(t) \wedge D(u,as).acc \subset\subset D(t,as).acc)$. Since $\rightarrow_{ST}$ is deterministic, this implies that $u' \downarrow s$ and that $s \in L(u') \Rightarrow s \in L(t') \wedge D(u',s).acc \subset\subset D(t',s).acc$. Therefore $t'\ R\ u'$, and R is a $\langle \Pi, \emptyset \rangle$-prebisimulation.

($\Leftarrow$) Suppose that $t(p) \sqsubseteq_{\langle \Pi, \emptyset \rangle} t(q)$; we must show that for all $s \in (Act - \{\tau\})^*$, $p \downarrow s \Rightarrow q \downarrow s \wedge \mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$. Fix $s \in (Act - \{\tau\})^*$ and assume that $p \downarrow s$; it is sufficient to establish:

1. $q \downarrow s$, and
2. $s \in L(q) \Rightarrow (s \in L(p) \wedge \mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s))$.

Let s' be the longest prefix of s with $s' \in L(q)$. From the definition of $\downarrow$ it follows that $p \downarrow s'$, and Lemma 3.5(2.) guarantees that $t(p) \downarrow s'$. Since $\sqsubseteq_{\langle \Pi, \emptyset \rangle}$ is a $\langle \Pi, \emptyset \rangle$-prebisimulation, the deterministic nature of $\rightarrow_{ST}$ also guarantees that $D(t(p),s')$ exists and $D(t(p),s') \sqsubseteq_{\langle \Pi, \emptyset \rangle} D(t(q),s')$, meaning that $t(q) \downarrow s'$ and $D(t(q),s').acc \subset\subset D(t(p),s').acc$. From this, we can conclude that $q \downarrow s'$, and hence $q \downarrow s$; moreover, if $s' = s$, meaning that $s \in L(q)$, it follows (via Lemma 3.5(3.) and Lemma 2.11) that $\mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$. □

The may preorder has a somewhat simpler description in terms of WAgraphs.

Theorem 4.2. Let $p, q \in P$. Then $p \sqsubseteq_{may} q$ if and only if $d(p) \sqsubseteq_{\langle D \times D, D \times Act \rangle} d(q)$ in $\mathcal{D}(\mathcal{L})$.

Proof. ($\Rightarrow$) Suppose $p \sqsubseteq_{may} q$; we must define a $\langle D \times D, D \times Act \rangle$-prebisimulation R relating $d(p)$ and $d(q)$. Define R as follows.

$$R = \{\ \langle d, e \rangle \mid L(d) \subseteq L(e)\ \}$$

It is straightforward to verify that this relation has the desired properties.

($\Leftarrow$) Assume that $d(p) \sqsubseteq_{\langle D \times D, D \times Act \rangle} d(q)$, and suppose that $s \in L(p)$. That

$s \in L(q)$ follows from the fact that $\sqsubseteq_{\langle D \times D, D \times Act \rangle}$ is a $\langle D \times D, D \times Act \rangle$-prebisimulation. □

The Equivalence Relations

The characterizations of the equivalence relations are the following.

Theorem 4.3. Let $p, q \in P$, and define $\Pi = \{ \langle t, u \rangle \in T \times T \mid t.acc = u.acc \}$. Then the following hold.

1. $p =_{may} q$ if and only if $d(p) \sim_{D \times D} d(q)$ in $\mathcal{D}(\mathcal{L})$.
2. $p =_{must} q$ if and only if $t(p) \sim_{\Pi} t(q)$ in $\mathcal{ST}(\mathcal{L})$.
3. $p =_{t} q$ if and only if $t(p) \sim_{\Pi} t(q)$ in $\mathcal{T}(\mathcal{L})$.

Proof. We shall prove the last of these; the other two cases are similar and are left to the reader.
($\Rightarrow$) Assume that $p =_t q$. To show that $t(p) \sim_{\Pi} t(q)$ it is sufficient to exhibit a Π-bisimulation R relating $t(p)$ and $t(q)$. Define relation R as follows.

$$\begin{aligned} R \quad = \quad & \{ \langle t, u \rangle \mid L(t) = L(u) \wedge \\ & \forall s \in (Act - \{\tau\})^*. \\ & (t \downarrow s \iff u \downarrow s) \wedge ((t \downarrow s \wedge s \in L(t)) \Rightarrow D(t, s).acc = D(u, s).acc) \} \end{aligned}$$

To establish that R is the required relation we first show that $t(p)\ R\ t(q)$. Since $p =_t q$ it follows that $p =_{may} q$, whence $L(p) = L(q)$. On the basis of Lemma 3.4(1.), then, we may conclude that $L(t(p)) = L(t(q))$. Now fix $s \in (Act - \{\tau\})^*$. Again, the fact that $p =_t q$ allows us to conclude that $p \downarrow s$ if and only if $q \downarrow s$, and hence, by Lemma 3.5(2.), $t(p) \downarrow s$ if and only if $t(q) \downarrow s$. Now assume that $t(p) \downarrow s$ and $s \in L(t(p))$. It follows that $t(q) \downarrow s$ and $s \in L(t(q))$, whence $p \downarrow s$ and $q \downarrow s$. From the definition of testing equivalence, we have that $\mathcal{A}(p, s) \subset\subset \mathcal{A}(q, s)$ and $\mathcal{A}(q, s) \subset\subset \mathcal{A}(p, s)$, and by Lemma 3.5(3.) and Lemma 2.11(2.) we may conclude that $D(t(p), s).acc = D(t(q), s).acc$.

We now must prove that R is a Π-bisimulation. To this end, suppose that $t\ R\ u$. From the definition of R and the fact that $t \uparrow$ if and only if $t.acc = \emptyset$ it follows that $\langle t, u \rangle \in \Pi$. Now suppose that $t \xrightarrow{a}_T t'$; we must show that there is a u' such that $u \xrightarrow{a}_T u'$ and $t'\ R\ u'$. Since Agraphs are deterministic, t' is unique, and since $t \xrightarrow{a}_T t'$ it follows that $a \in L(t)$, whence $a \in L(u)$ by the definition of R. So there is a unique u' with $u \xrightarrow{a}_T u'$. We now must show that $t'\ R\ u'$. Since $L(t) = L(u)$ and t' and u' are unique, it is clearly the case that $L(t') = L(u')$. Now fix $s \in (Act - \{\tau\})^*$. Since $t \downarrow as$ if and only if $u \downarrow as$, and since t' and u' are unique, it follows that $t' \downarrow s$ if and only if $u' \downarrow s$. Now suppose that $t' \downarrow s$ and $s \in L(t')$. These imply that $t \downarrow as$ and $as \in L(t)$, meaning that $D(t, as).acc = D(u, as).acc$. As $D(t', s).acc = D(t, as).acc$ and $D(u', s).acc = D(u, as).acc$, we have that $D(t', s).acc = D(u', s).acc$, and we have established that $t'\ R\ u'$. Arguing symmetrically, we can establish that if $u \xrightarrow{a}_T u'$ then there is a t' with $t \xrightarrow{a}_T t'$ and $t'\ R\ u'$, and R is therefore a Π-bisimulation.

($\Leftarrow$) Assume that $t(p) \sim_{\Pi} t(q)$; we must show that $p =_t q$, or equivalently, that $p =_{may} q$ and $p =_{must} q$. The former is simple to verify; we concentrate on

the latter. To prove this, we are required to establish that $p \sqsubseteq_{must} q$ and $q \sqsubseteq_{must} p$; this is equivalent to showing the following.

$$\forall s \in (Act - \{\tau\})^*. \quad (p \downarrow s \iff q \downarrow s) \wedge (p \downarrow s \Rightarrow (\mathcal{A}(p,s) \subset\subset \mathcal{A}(q,s) \wedge \mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)))$$

Fix $s \in (Act - \{\tau\})^*$. It is easy to show that $t(p) \downarrow s$ if and only if $t(q) \downarrow s$ from the fact that $t(p) \sim_\Pi t(q)$, and by Lemma 3.5(2.) it follows that $p \downarrow s$ if and only if $q \downarrow s$. Now suppose that $p \downarrow s$. If $s \notin L(p)$ then $s \notin L(q)$, and this means that $\mathcal{A}(p,s) = \mathcal{A}(q,s) = \emptyset$, so the acceptance set condition is trivially satisfied. Now suppose that $s \in L(p)$. Then $s \in L(q)$, and from Lemma 3.4(1.) it follows that $s \in L(t(p))$ and $s \in L(t(q))$. From the fact that $t(p) \sim_\Pi t(q)$ it follows that $D(t(p),s).acc = D(t(q),s).acc$, and the desired result is a consequence this fact and Lemmas 3.5(3.) and 2.11(1.). □

5. Deciding Testing for Finite-State Processes

In Section 2 we showed how, given a labeled transition system $\langle P, Act, \rightarrow \rangle$ and a state $p \in P$, one can define another, smaller, labeled transition corresponding to the process with "start state" p. These were written $\langle P_p, Act, \rightarrow_p \rangle$, where each state in P_p is assumed to be reachable from p. In this section we consider the problem of deciding the testing relations for such processes when $|\rightarrow_p|$ is finite.

We examine the equivalence relations first, since there are well-known algorithms for computing bisimulation equivalence [KaS90, PaT87]. These algorithms work in the following way, for transition systems $\langle P_p, Act, \rightarrow_p \rangle$ and $\langle P_{p'}, Act, \rightarrow_{p'} \rangle$ where P_p and $P_{p'}$ are assumed to be disjoint.

1. Form $\langle P_p \cup P_{p'}, Act, \rightarrow_p \cup \rightarrow_{p'} \rangle$.
2. Starting with the universal relation $(P_p \cup P_{p'}) \times (P_p \cup P_{p'})$, apply a relation refinement procedure to generate the coarsest bisimulation.
3. If p and p' are related by this bisimulation then the transition systems are equivalent; otherwise, they are not.

To use these algorithms to compute the testing equivalences, they first need to be modified to compute Π-bisimulations. In the case when Π is an equivalence relation, this is straightforward—replace the reference to the universal relation in step (2.) with a reference to Π. Note that for the testing equivalences, Π is always an equivalence relation.

The other step in computing the equivalences involves defining procedures for computing appropriate Agraphs; given a transition system $\langle P_p, Act, \rightarrow \rangle$, the Agraphs in question are the portions of $\mathcal{T}(\langle P, Act, \rightarrow \rangle)$, $\mathcal{ST}(\langle P, Act, \rightarrow \rangle)$, and $\mathcal{D}(\langle P, Act, \rightarrow \rangle)$ reachable from $t(p)$, $t(p)$ and $d(p)$, respectively, with "start states" $t(p)$, $t(p)$ and $d(p)$. We shall abuse notation slightly by referring to these Agraphs as $\mathcal{T}(\langle P_p, Act, \rightarrow \rangle)$, $\mathcal{ST}(\langle P_p, Act, \rightarrow \rangle)$ and $\mathcal{D}(\langle P_p, Act, \rightarrow \rangle)$, respectively. Figure 2 sketches an algorithm for computing $\mathcal{T}(\langle P_p, Act, \rightarrow \rangle)$. The method used is a variant of one presented in [HoU79] for generating deterministic automata from nondeterministic ones. Minor variations on this procedure compute $\mathcal{ST}(\langle P_p, Act, \rightarrow \rangle)$ and $\mathcal{D}(\langle P_p, Act, \rightarrow \rangle)$. Figure 3 gives examples of the graphs produced by these transformations.

These transformations are in general PSPACE-complete, since the problem

Assume:

- $E : (2^P \times \mathit{bool}) \rightarrow \mathit{Astate}$ is an environment (partial function), initially everywhere undefined, mapping pairs comprising a set of states and a boolean to a Agraph state;
- *newstate* is a function that allocates a new Agraph state, given an acceptance set and a boolean.

Function *build* takes a transition system, a partially constructed Agraph, a set of states and a boolean and returns a pair consisting of a Agraph and its "start state".

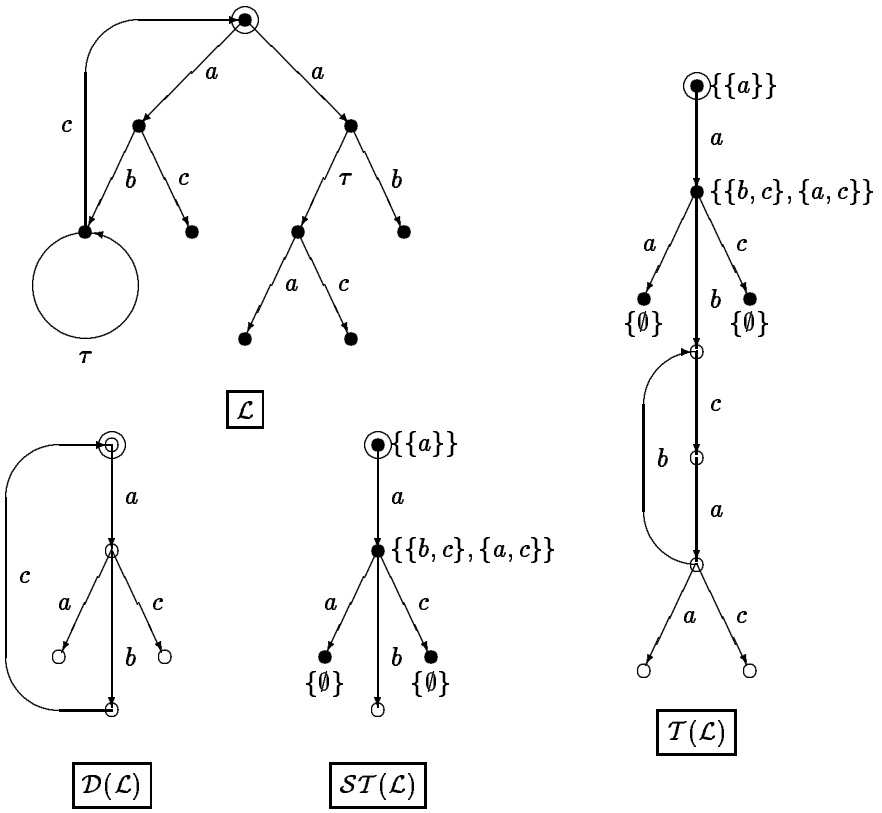
```
fun build (⟨P, Act, →⟩, ⟨T, Act, →_T⟩, S ⊆ P, b): Agraph × Astate =
    Construct S^τ;
    if E(⟨S^τ, b⟩) exists then return ⟨⟨T, Act, →_T⟩, E(⟨S^τ, b⟩)⟩
    else let c = (b ∧ ∀p ∈ S^τ . p↓ε)
             acc = if c then { S(p) | p ∈ S^τ ∧ p -τ↛ } else ∅
             t = newstate(acc, c)
         in
             E := E[⟨S^τ, c⟩ ↦ t];
             T := T ∪ {t};
             foreach a ∈ { a ∈ Act − {τ} | ∃p ∈ S^τ . s -a→ } do
                 let S_a = D(S^τ, a)
                     ⟨L, t'⟩ = build(⟨P, Act, →⟩, ⟨T, Act, →_T⟩, S_a, c)
                 in →_T := →_T ∪{⟨t, a, t'⟩} end;
             return ⟨⟨T, Act, →_T⟩, t⟩
         end;

fun T(⟨P_p, Act, →⟩) = build(⟨P_p, Act, →⟩, ⟨∅, Act, ∅⟩, {p}, true);
```

Fig. 2. A procedure for computing $\mathcal{T}$.

of computing failures equivalence (which is the same as testing equivalence) is known to be PSPACE-complete and the bisimulation algorithms are of polynomial complexity [KaS90, PaT87]. Intuitively, the difficulty arises from the fact that a Agraph might have two states corresponding to each subset of states in the original graph. Practice, however, indicates that Agraphs in fact have *fewer* states than the transition systems from which they are generated, owing to the fact that $\xrightarrow{\tau}$-transitions are collapsed. Our techniques have been implemented in the Concurrency Workbench [CPS89, CPS90, CPS92], and our experience for a variety of problems has been that computing testing equivalence takes about twice as long as computing observational equivalence.

Computing the preorder relations can be performed in much the same way as calculating the equivalences. First, modify a "refinement-style" prebisimulation algorithm to compute $\langle \Pi, \Psi \rangle$-prebisimulations, and then apply the procedure to the appropriate deterministic graphs. The interested reader is referred to [CPS89, CPS90, CPS92].



"Start" nodes are circled. Closed nodes are represented as filled circles, open nodes as unfilled. Acceptance sets have been left off open nodes.

Fig. 3. Examples of the transition system transformations.

6. An Example

In this section we consider an example illustrating our techniques for computing the testing relations. The systems that we investigate are based on ones proposed by Nigel Edwards of HP Labs and refined by Luca Aceto at Sussex University for modeling part of an office automation system.

Figure 4 describes $SPEC$, the specification of the desired behavior of a system in which two users compete for a shared resource. Initially, either user (but not both) may issue its *hold* command, thereby gaining exclusive access to the resource. This user may then nondeterministically choose either to execute an operation or to free the resource.

We now examine an implementation that we wish to assert as correctly implementing this system, in the sense of being testing equivalent to $SPEC$. The structure of the system appears in Figure 5 and consists of five components: two users, two one-place buffers, and the shared resource (or object). The buffers mediate the communication between the users and the resource; each users issues

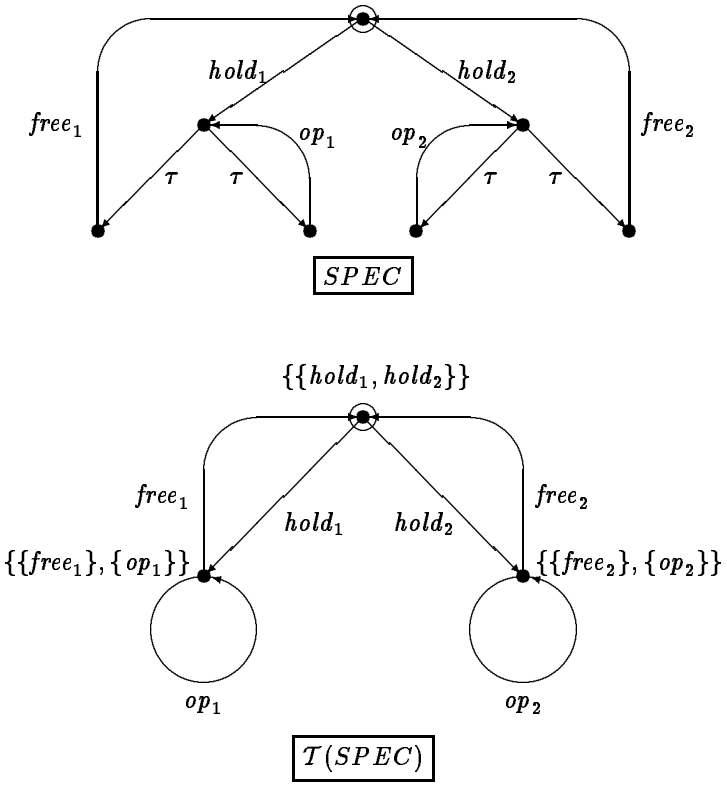


Fig. 4. The transition system for $SPEC$ and its associated Agraph.

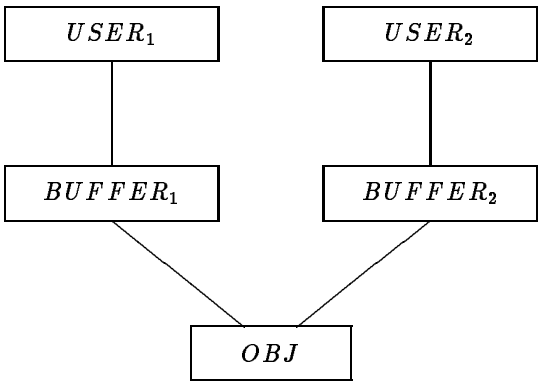


Fig. 5. The structure of IMP.

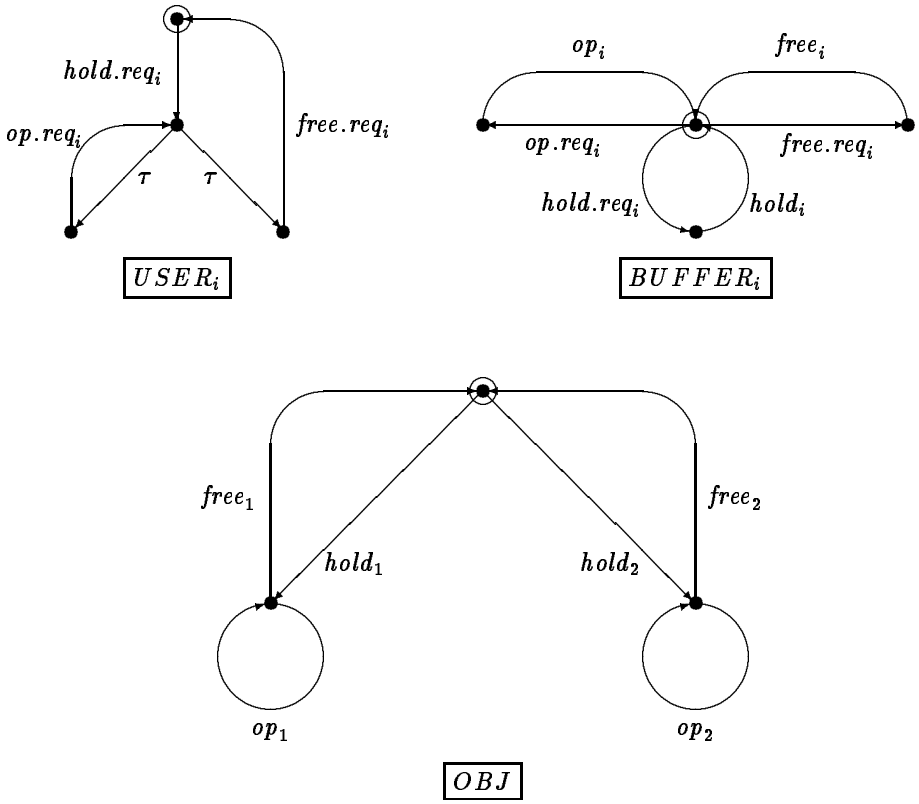


Fig. 6. The transition systems for $USER_i$, $BUFFER_i$ and OBJ.

its commands to the appropriate buffer, which then relays them to the object. The transition systems corresponding to these processes appear in Figure 6.

To assemble the final implementation, IMP, from the components described in Figure 6 we use operators taken from CSP. They are briefly described below; the interested reader is referred to [Hoa85] for a more detailed account. We first need to define the *sort* of a state p in a labeled transition system; it is given as follows.

$$sort\, p = (\bigcup\{\, S(p') \mid \exists s \in (Act - \{\tau\})^*.\ p \stackrel{s}{\Rightarrow} p' \,\}) - \{\tau\}$$

So the sort of a state is the set of visible actions that may be performed during executions starting from state p.

The first two operators define different notions of parallel composition. Suppose that $\mathcal{L}_1 = \langle P_p, Act, \rightarrow_p \rangle$ and $\mathcal{L}_2 = \langle Q_q, Act, \rightarrow_q \rangle$ are two transition systems with start states p and q, respectively. Then $\mathcal{L}_1 || \mathcal{L}_2$ is the transition system $\langle (P \times Q)_{\langle p,q \rangle}, Act, \rightarrow^{p||q} \rangle$, where $\rightarrow^{p||q}$ is defined as follows.

$$p_1 \stackrel{a}{\rightarrow} p_2, a \notin (sort\, p) \cap (sort\, q) \quad \Rightarrow \quad \langle p_1, q_1 \rangle \stackrel{a}{\rightarrow}^{p||q} \langle p_2, q_1 \rangle$$

$$q_1 \stackrel{a}{\rightarrow} q_2, a \notin (sort\, p) \cap (sort\, q) \quad \Rightarrow \quad \langle p_1, q_1 \rangle \stackrel{a}{\rightarrow}^{p||q} \langle p_1, q_2 \rangle$$

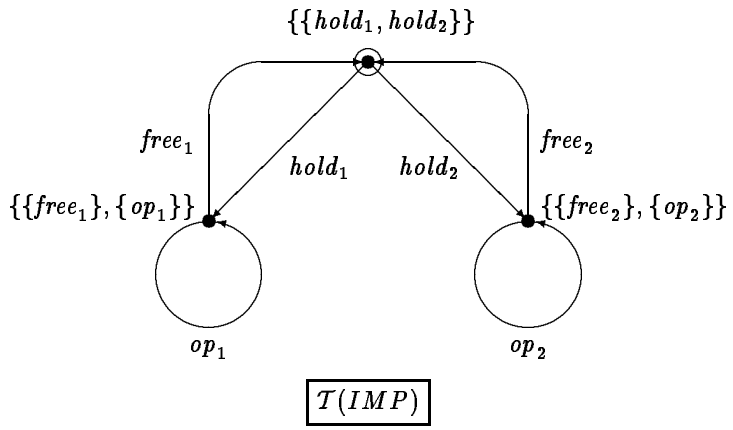


Fig. 7. The Agraph for IMP.

$$p_1 \stackrel{a}{\rightarrow} p_2, q_1 \stackrel{a}{\rightarrow} q_2, a \in (sort\, p) \cap (sort\, q) \quad \Rightarrow \quad \langle p_1, q_1 \rangle \stackrel{a}{\rightarrow}^{p||q} \langle p_2, q_2 \rangle$$

Notice that a state $\langle p', q' \rangle$ may perform an action a that is in the sort of both p and q exactly when both p' and q' are both capable of performing the action; other actions are executed in an interleaved fashion. The second operator is a strict interleaving operator; $\mathcal{L}_1 ||| \mathcal{L}_2$ defines the transition system $\langle (P \times Q)_{\langle p,q \rangle}, Act, \rightarrow^{p|||q} \rangle$, with $\rightarrow^{p|||q}$ defined by the following.

$$p_1 \stackrel{a}{\rightarrow} p_2 \quad \Rightarrow \quad \langle p_1, q_1 \rangle \rightarrow^{p|||q} \langle p_2, q_1 \rangle$$
$$q_1 \stackrel{a}{\rightarrow} q_2 \quad \Rightarrow \quad \langle p_1, q_1 \rangle \rightarrow^{p|||q} \langle p_1, q_2 \rangle$$

The third operator is a *hiding operator*; given a transition system $\mathcal{L} = \langle P_p, Act, \rightarrow \rangle$ and a set $C \subseteq Act$, the transition system defined by $\mathcal{L} \backslash C$ is $\mathcal{L}$ with every transition with label $c \in C$ "relabeled" to τ.

Given these definitions, we may define IMP as follows. Let C_i, $i \in \{1, 2\}$, be the set $\{hold.req_i, free.req_i, op.req_i\}$.

$$IMP = ((USER_1 || BUFFER_1) \backslash C_1 \ ||| \ (USER_2 || BUFFER_2) \backslash C_2) \ || \ OBJ$$

The resulting transition system has 37 states. The associated Agraph is shown in Figure 7.

In light of the results in this paper, in order to show that $IMP =_t SPEC$, it suffices to show that the Agraphs in Figures 4 and 7 are Π-bisimilar, where $\langle s_1, s_2 \rangle \in \Pi$ exactly when $s_1.acc = s_2.acc$. A quick inspection in fact shows that the two Agraphs are isomorphic, and hence $IMP =_t SPEC$. It is interesting to note that it is *not* the case that IMP and $SPEC$ are observationally equivalent in the sense of [Mil89]. To see why, it is sufficient to exhibit a formula in the Hennessy-Milner logic [HeM85] that $SPEC$ satisfies and IMP does not. One such formula is the following.

$$[hold_1](\langle op_1 \rangle \langle op_1 \rangle \, True \vee \langle free_1 \rangle \, True)$$

A state s satisfies this formula if for every s' such that $s \stackrel{hold_1}{\Rightarrow} s'$, $op_1 op_1 \in L(s')$ or $free_1 \in L(s')$. The start state of $SPEC$ clearly satisfies this formula, while the start state of IMP does not. The reason for the latter is in IMP, after

$hold_1$ has been executed, it is possible that $USER_1$ buffers an op_1 request and then nondeterministically decides that its next request will be a $free_1$. In this situation, even though the only visible action that has been executed is a $hold_1$, it has already been determined that the next two visible actions will be op_1 and $free_1$.

7. Conclusions

In this paper we have presented a bisimulation/prebisimulation-based characterization of the testing relations that relies on transformations of the underlying model of processes. As a result, it is possible to use existing algorithms for computing bisimulations [KaS90, PaT87] and prebisimulations [CPS89, CPS90, CPS92] in conjunction with appropriate transformation procedures to compute the testing equivalences and preorders. Although the problem of computing these relations is known to be PSPACE-complete [KaS90], experience with the implementation in the Concurrency Workbench [CPS89, CPS90, CPS92] suggests that in practice they are computationally well-behaved.

Other equivalences and preorders can also be characterized in terms of bisimulations and prebisimulations on appropriately transformed labeled transition systems. For example, *observational equivalence* and *observational congruence* [Mil89] may both be defined in this way, as can *GSOS congruence* [BIM88] and the $\frac{2}{3}$-*bisimulation* preorder and equivalence [LaS89]. This suggests that our technique of using general bisimulation and prebisimulation algorithms on transformed transition systems may be applied to compute other relations, in addition to the testing ones.

References

[BIM88] Bloom, B., Istrail, S., and Meyer, A. "Bisimulation Can't Be Traced." Proceedings of the ACM Symposium on Principles of Programming Languages, 1988.

[CPS89] Cleaveland, R., Parrow, J., and Steffen, B. "The Concurrency Workbench." In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, pp. 24–37. *Lecture Notes in Computer Science* 407. Springer-Verlag, Berlin, 1989.

[CPS90] Cleaveland, R., Parrow, J., and Steffen, B. "A Semantics-Based Verification Tool for Finite-State Systems." In *Proceedings of the Ninth Annual Workshop on Protocol Specification, Testing and Verification*, pp. 287–302. North-Holland, Amsterdam, 1990.

[CPS92] Cleaveland, R., Parrow, J., and Steffen, B. "The Concurrency Workbench: A Semantics-Based Verification Tool for Finite-State Systems." To appear in *ACM Transactions on Programming Languages and Systems.*

[DeH84] DeNicola, R. and Hennessy, M. "Testing Equivalences for Processes." *Theoretical Computer Science* 24, 1984, pp. 83–113.

[Hen81] Hennessy, M. "A Term Model for Synchronous Processes." *Information and Control*, v. 51, n. 1, October 1981, pp. 58–75.

[Hen85] Hennessy, M. "Acceptance Trees." *Journal of the ACM*, v. 32, n. 4, October 1985, pp. 896–928.

[Hen88] Hennessy, M. *Algebraic Theory of Processes.* MIT Press, Cambridge, 1988.

[HeM85] Hennessy, M. and Milner, R. "Algebraic Laws for Nondeterminism and Concurrency." *Journal of the ACM*, v. 32, n. 1, January 1985, pp. 137–161.

[Hoa85] Hoare, C.A.R. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[HoU79] Hopcroft, J. and Ullman, J. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, 1979.

[KaS90] Kanellakis, P.C. and Smolka, S.A. "CCS Expressions, Finite State Processes, and Three Problems of Equivalence." *Information and Computation*, v. 86, n. 1, May 1990, pp. 43–68.

[LaS89] Larsen, K. and Skou, A. "Bisimulation through Probabilistic Testing." Proceedings of the ACM Symposium on Principles of Programming Languages, 1989.

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

[Mil83] Milner, R. "Calculi for Synchrony and Asynchrony." *Theoretical Computer Science*, v. 25, n. 3, July 1983, pp. 267–310.

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

[PaT87] Paige, R. and Tarjan, R.E. "Three Partition Refinement Algorithms." *SIAM Journal of Computing*, v. 16, n. 6, December 1987, pp. 973–989.

[Par81] Park, D. "Concurrency and Automata in Infinite Strings." Lecture Notes in Computer Science 104, pp. 167–183. Springer-Verlag, Berlin, 1981.

[Wal88] Walker, D. "Bisimulations and Divergence in CCS." Proceedings of the Third Annual Symposium on Logic in Computer Science, pp. 186–192. Computer Society Press, 1988.