Testing Equivalence as a Bisimulation Equivalence*

Rance Cleaveland
Computer Science Department
North Carolina State University
Box 8206
Raleigh, NC 27695
USA

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

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. Using 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 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* [13, 15, 16] 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 [9, 14]. However, it is often the case that

*Research supported by British Science and Engineeering Research Council grant GC/D69464.

these relations are too discriminating; two processes may be deemed unrelated even though there is no practical way of ascertaining this to be the case. It is also the case that there is no smooth connection between the preorders and the equivalences.

2. *Testing/Failures* [3, 5, 7] 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 a notion of *process transformation*. 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 [2], 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 [9], 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 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 lists 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 $p \xrightarrow{a} p'$ in lieu of $\langle p, a, p' \rangle \in \rightarrow$, and we shall on occasion use $p \xrightarrow{a}$ to mean that there is a p' with $p \xrightarrow{a} p'$ 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 $p \in P$, $|\{ p' \mid \exists a.\, p \xrightarrow{a} p' \}| < \infty$.

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 it 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

Let $\langle P, Act, \rightarrow \rangle$ be an arbitrary labeled transition system. *Bisimulation equivalence*, $\sim$, is an equivalence relation between elements of P that relates processes on the basis of their behaviour. It is defined in terms of relations called *bisimulations*. Here we shall define a mild generalization of bisimulation equivalence.

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. $p \xrightarrow{a} p' \Rightarrow \exists q'.\, q \xrightarrow{a} q' \wedge p'Rq'$.
2. $q \xrightarrow{a} q' \Rightarrow \exists p'.\, p \xrightarrow{a} p' \wedge p'Rq'$.

Notice that if $\Pi = P \times P$ then a Π-bisimulation is a bisimulation in the usual sense [12, 13, 15].

The relation $\sim_\Pi$ is defined in terms 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*, $\sqsubseteq$, is a behavioural preorder defined on processes that is defined in terms of prebisimulations [16]. 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 $\neg(p \Uparrow a)$. We shall define a slight generalization of the usual notion.

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. $\langle q, a \rangle \in \Psi \Rightarrow [\langle p, a \rangle \in \Psi \wedge (p \xrightarrow{a} p' \Rightarrow \exists q'.\, q \xrightarrow{a} q' \wedge p'Rq')]$.

2. $p \Downarrow a \Rightarrow [q \Downarrow a \wedge (q \xrightarrow{a} q' \Rightarrow \exists p'.\ p \xrightarrow{a} p' \wedge p' R q')]$.

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. Note that if $\Pi = P \times P$ and $\Psi = P \times Act$ then $\sqsubseteq_{\langle \Pi, \Psi \rangle}$ coincides with $\sqsubseteq$.

2.2 The Testing Relations

The testing relations are defined on the basis of two preorders, $<<_{may}$ and $<<_{must}$ [3, 5]. These may be characterized in terms of the responses of processes to a collection of tests; we shall however use an alternative characterization that relies on the following definitions.

Definition 2.6 *Let $s, s' \in (Act - \{\tau\})^*$ be sequences of visible actions, and let $a \in Act$.*

1. *$\stackrel{s}{\Rightarrow}$ is defined inductively on the structure of s as follows.*
   (a) *$\stackrel{\epsilon}{\Rightarrow} = \xrightarrow{\tau}^*$, where $\xrightarrow{\tau}^*$ is the transitive and reflexive closure of $\xrightarrow{\tau}$.*
   (b) *$\stackrel{as'}{\Rightarrow} = \stackrel{\epsilon}{\Rightarrow} \circ \xrightarrow{a} \circ \stackrel{s'}{\Rightarrow}$, 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.*
3. *$S(p) = \{\, a \in Act \mid p \xrightarrow{a} \,\}$.*
4. *$D(p, a) = \{\, p' \mid p \xrightarrow{a} p' \,\}$; $D(p) = \bigcup_{a \in Act} D(p, a)$.*
5. *The* convergence relation *$p \downarrow s$ is defined inductively as follows.*
   (a) *$p \downarrow \epsilon$ if and only if there is no infinite sequence $\langle p_i \rangle_{i \geq 0}$ such that $p \xrightarrow{\tau} p_0$ and $p_i \xrightarrow{\tau} p_{i+1}$.*
   (b) *$p \downarrow as'$ if and only if $p \downarrow \epsilon$ and $p \stackrel{a}{\Rightarrow} p'$ implies $p' \downarrow s'$.*
6. *The* acceptance set *of p after s is defined as follows.*

$$\mathcal{A}(p, s) = \{\, S(p') \mid p \stackrel{s}{\Rightarrow} p' \wedge p' \stackrel{\tau}{\not\rightarrow} \,\}$$

Most of these notions are straightforward. Notice that our assumption about finite-branching ensures that $D(p)$ is always finite; it also implies that when $p \downarrow s$, $\mathcal{A}(p, s)$ is a finite set of finite sets. Also, $p \downarrow s$ if it is impossible for p to perform an infinite sequence of τ actions during the "execution" of s; it does *not* imply that $s \in L(p)$. The acceptance set $\mathcal{A}(p, s)$ represents the set of "action capabilities" of p after s. A set S is in $\mathcal{A}(p, s)$ if it is possible for a state to be reached from p via s such that the only next possible actions are exactly those in S. 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.

We also define an inclusion relation on sets of sets of actions in the following way.

Definition 2.7 *Let $A, B \subseteq 2^{Act}$. Then $A \subset\subset B$ if it is the case that:*

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

This relation is used in conjunction with acceptance sets to capture a notion of "less nondeterministic than".

We may now define $<<_{may}$ and $<<_{must}$ as follows.

Definition 2.8

1. *$p <<_{may} q$ if $L(p) \subseteq L(q)$.*
2. *$p <<_{must} q$ if $\forall s \in (Act - \{\tau\})^*.\ p{\downarrow}s \Rightarrow (q{\downarrow}s \wedge \mathcal{A}(q, s) \subset\subset \mathcal{A}(p, s))$.*

The $<<_{may}$ relation corresponds to language containment, while $<<_{must}$ 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, $p{\downarrow}s$ – $p <<_{must} q$ implies that $q <<_{may} p$.

It also turns out that $<<_{must}$ can be defined with respect to "minimized" version of acceptance sets. Consider the operator *min* on acceptance sets defined by

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

We then have the following result.

Lemma 2.9 *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 $min\ A = min\ B$.*

The other testing relations are defined as follows.

Definition 2.10 *The testing preorder, $<<_t$, and the testing equivalences, $=_{may}, =_{must}$ and $=_t$, are defined by:*

1. *$p <<_t q$ if $p <<_{may} q$ and $p <<_{must} q$, and*
2. *$p =_i q$ if $p <<_i q$ and $q <<_i p$, for $i \in \{may, must, t\}$.*

3 Tgraphs, STgraphs and Dgraphs

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 $\xrightarrow{\tau} = \emptyset$ 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.

Tgraphs are a particular class of labeled transition systems satisfying certain properties.

Definition 3.2 *A labeled transition system $\langle T, Act, \rightarrow\rangle$ is a Tgraph if the following hold.*

1. *$\rightarrow$ is deterministic.*

2. *Each $t \in T$ is labeled by two pieces of information, $t.acc$ and $t.closed$, with $t.acc \subseteq 2^{Act}$ and $t.closed$ a boolean.*

3. *$|t.acc| < \infty$, $\forall S \in t.acc.\ |S| < \infty$, and $t.acc = min\, t.acc$.*

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 Tgraph should be compared with notion of acceptance tree in [4, 5]. Intuitively, $t.acc$ is an acceptance set, while $t.closed$ represents whether or not a node is "convergent" (or *closed*, in the terminology of [4]). Notice that if $t.closed$ holds, then $t.closed$ must hold of each predecessor of t along $\rightarrow$. On occasion, we shall say that a Tgraph node t is *closed* if $t.closed$ is true, and *open* otherwise.

Two variants of Tgraphs are also of interest.

Definition 3.3

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

2. *Tgraph $\langle T, Act, \rightarrow\rangle$ is a Dgraph if for each $t \in T$, $t.closed = false$.*

3.1 Building Tgraphs

We now define a construction for generating Tgraphs from arbitrary labeled transition systems; these Tgraphs turn out to enjoy certain "language equivalence" properties with the graphs from which they are built. We first define the τ-*closure* of a set of states Q as follows.

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

Notice that $Q \subseteq Q^\tau$ for any Q. Also, extend the definitions of D and $\downarrow s$ to sets of states:

$$\begin{array}{rcl} D(Q,a) & = & \bigcup_{q \in Q} D(q,a) \\ Q \downarrow s & \iff & \bigwedge_{q \in Q} q \downarrow s \end{array}$$

Now let $\mathcal{L} = \langle P, Act, \rightarrow \rangle$ be a labeled transition system. The Tgraph $\mathcal{T}(\mathcal{L}) = \langle T, Act, \rightarrow_T \rangle$ is given as follows.

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

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

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) $(D(Q_1, a))^\tau = Q_2$.
   (c) $[b_1 = \mathit{false} \Rightarrow b_2 = \mathit{false}] \wedge [(b_1 = \mathit{true} \wedge b_2 = \mathit{false}) \Rightarrow Q_2 \uparrow \epsilon]$.

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

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

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

3.2 Properties of Tgraphs

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 $p \in P$.

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

$t(p)$ 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 Tgraphs, 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 Tgraphs in the following way.

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

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

1. *$L(p) = L(t(p))$, where $L(t(p))$ is interpreted with respect to $\mathcal{T}(\mathcal{L})$.*
2. *$L(p) = L(d(p))$, where $L(d(p))$ is interpreted with respect to $\mathcal{D}(\mathcal{L})$.*

In general, this property does not hold between p and its corresponding state in $\mathcal{ST}(\mathcal{L})$. We do have the following.

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

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

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 various Tgraphs. Let $\mathcal{L} = \langle P, Act, \rightarrow \rangle$ be an arbitrary labeled transition system. We shall use the following notation.

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

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 Tgraphs. We use the following.

$$t \Downarrow a \iff t.closed = true$$

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

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

1. $p \ll_{must} q$ *if and only if* $t(p) \mathrel{\sqsubset\!\!\!\sim}_{\langle \Pi, \emptyset\rangle} t(q)$ *in* $\mathcal{ST}(\mathcal{L})$*.*

2. $p \ll_{t} q$ *if and only if* $t(p) \mathrel{\sqsubset\!\!\!\sim}_{\langle \Pi, T \times Act\rangle} t(q)$ *in* $\mathcal{T}(\mathcal{L})$*.*

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

Theorem 4.2 *Let* $p, q \in P$*. Then* $p \ll_{may} q$ *if and only if* $d(p) \mathrel{\sqsubset\!\!\!\sim}_{\langle D\times D, D\times Act\rangle} d(q)$ *in* $\mathcal{D}(\mathcal{L})$*.*

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* $\mathcal{T}(\mathcal{L})$*.*

5 Deciding Testing for Finite-State Processes

In section 2 we introduced the labeled transition systems corresponding to the process with "start state" p taken from a larger transtion system. 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 [9, 14]. These algorithms work in the following way, for transition systems $\langle P_p, Act, \rightarrow_p\rangle$ and $\langle P_{p'}, Act, \rightarrow_{p'}\rangle$.

1. Form $\langle P_p + P_{p'}, Act, \rightarrow_p + \rightarrow_{p'}\rangle$, where $+$ denotes disjoint union.

2. Starting with the universal relation $(P_p + P_{p'}) \times (P_p + P_{p'})$, 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.

Assume:

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

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

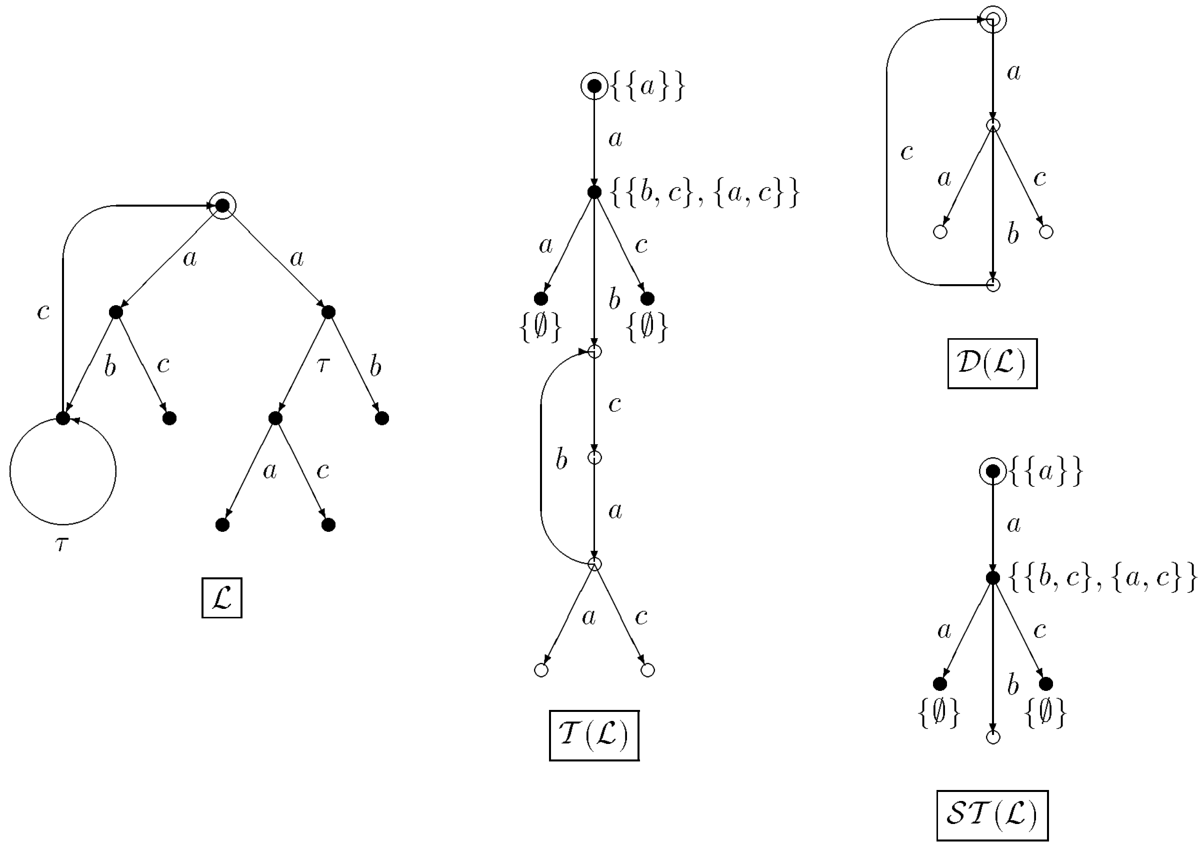
fun *build* $(\langle P, Act, \rightarrow\rangle,\ \langle T, Act, \rightarrow_T\rangle,\ S \subseteq P,\ b)$: $Tgraph \times Tstate =$
  Construct S^τ;
  if $E(\langle S^\tau, b\rangle)$ *exists* then return $\langle\langle T, Act, \rightarrow_T\rangle, E(\langle S^\tau, b\rangle)\rangle$
  else let $c = (b \wedge \forall p \in S^\tau.\ p{\downarrow}\epsilon)$
      $acc =$if c then $\{\, S(p) \mid p \in S^\tau \wedge p \overset{\tau}{\not\rightarrow} \,\}$ else $\emptyset$
      $t = newstate(acc, c)$
    in
      $E := E[\langle S^\tau, c\rangle \mapsto t]$;
      $T := T \cup \{t\}$;
      foreach $a \in \{\, a \in Act - \{\tau\} \mid \exists p \in S^\tau.\ s \overset{a}{\rightarrow} \,\}$ do
        let $S_a = D(S^\tau, a)$
          $\langle \mathcal{L}, t'\rangle = build(\langle P, Act, \rightarrow\rangle, \langle T, Act, \rightarrow_T\rangle, S_a, c)$
        in $\rightarrow_T\ :=\ \rightarrow_T \cup\{\langle t, a, t'\rangle\}$ end;
      return $\langle\langle T, Act, \rightarrow_T\rangle, t\rangle$
    end;

fun $\mathcal{T}(\langle P_p, Act, \rightarrow\rangle) = build(\langle P_p, Act, \rightarrow\rangle,\ \langle\emptyset, Act, \emptyset\rangle,\ \{p\},\ true)$;

Figure 1: A procedure for computing $\mathcal{T}$.

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 Tgraphs; given a transition system $\langle P_p, Act, \rightarrow\rangle$, the Tgraphs 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 and refer to these Tgraphs as $\mathcal{T}(\langle P_p, Act, \rightarrow\rangle)$, $\mathcal{ST}(\langle P_p, Act, \rightarrow\rangle)$ and $\mathcal{D}(\langle P_p, Act, \rightarrow\rangle)$. Figure 1 sketches an algorithm for computing $\mathcal{T}(\langle P_p, Act, \rightarrow\rangle)$. The method used is a variant of one presented in [8] for generating deterministic automata from



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

Figure 2: Examples of the transition system transformations.

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 2 gives examples of the graphs produced by these transformations.

These transformations are in general PSPACE-complete, since the problem 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. Intuitively, the difficulty arises from the fact that a Tgraph might have a state corresponding to each subset of states in the original graph and the fact that acceptance sets can theoretically have size exponential in the size of the action set. Practice, however, indicates that Tgraphs in fact have *fewer* states than the transition systems from which they are generated, owing to the fact that $\xrightarrow{\tau}$-transitions are collapsed. Moreover, acceptance sets are small in practice, since the size of the acceptance set indicates the degree of nondeterminism present in the process and this is usually small. Our techniques have been implemented in the Concurrency Workbench [2], 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. As these algorithms are not so well-known we shall say no more about them. The interested reader is referred to [2].

6 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 [9, 14] and prebisimulations [2] 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, experience with the implementation in the Concurrency Workbench [2] 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* [13] may both be defined in this way, as can *GSOS congruence* [1] and the $\frac{2}{3}$-*bisimulation* preorder and equivalence [10]. 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

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

[2] Cleaveland, R., J. Parrow and B. Steffen. "The Concurrency Workbench." To appear in the Proceedings of the Workshop on the Automated Verification of Finite-State Systems, Grenoble, 1989.

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

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

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

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

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

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

[9] Kanellakis, P.C. and S.A. Smolka. "CCS Expressions, Finite State Processes, and Three Problems of Equivalence." Proceedings of the ACM Symposium on Principles of Distributed Computing, 1983, pp. 228-240.

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

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

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

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

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

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

[16] 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.